VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Metodika a nástroj na ověření modelování procesů

Methods and Tool for Process Modeling Validation

2018 Bc. Tomáš Podolák

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Zadání diplomové práce

Student: **Bc. Tomáš Podolák**

Studijní program: N2647 Informační a komunikační technologie

Studijní obor: 2612T025 Informatika a výpočetní technika

Téma: Metodika a nástroj na ověření modelování procesů
Methods and Tool for Process Modeling Validation

Jazyk vypracování: čeština

Zásady pro vypracování:

Cílem diplomové je zpracování modelovací části metodiky a její implementace. Metodika se týká modelování, simulace a zpětné kontroly prováděných procesů. Nástroj bude, mimo jiné, podporovat zvolený metamodel, a dle něj nastavovat a ověřovat model pomocí pravidel a vizuálního modelování.

Práce bude obsahovat zejména:
1. Seznámení s problematikou a metodikou a současným stavem nástroje, dopracování dalších částí.
2. Podrobný popis modelovací části a její začlenění do nástroje.
3. Implementace nástroje.
4. Experimenty, vyhodnocení.

Seznam doporučené odborné literatury:

[1] John F. Sowa, Knowledge Representation: Logical, Philosophical, and Computational Foundations, Brooks Cole Publishing Co., Pacific Grove, CA, ©2000
[2] Alec Sharp, Patrick McDermott: Workflow Modeling: Tools for Process Improvement and Application Development, Artech House; 2 edition (October 31, 2008)
[3] Pfleeger, Shari Lawrence, and Joanne M. Atlee. 2009. Software Engineering: Theory and Practice: Prentice Hall, ISBN 0136061699
[4] Pressman, Roger S. 2010. Software Engineering : A Practitioner's Approach. 7th ed. New York: McGraw-Hill Higher Education, ISBN 9780073375977
[5] Sommerville, Ian. 2010. Software Engineering. 9th ed, International Computer Science Series. Harlow: Addison-Wesley, ISBN 978-0137035151

Další literatura podle pokynů vedoucího diplomové práce.

Formální náležitosti a rozsah diplomové práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí diplomové práce: **Ing. Svatopluk Štolfa, Ph.D.**

Datum zadání: 01.09.2017

Datum odevzdání: 30.04.2018

doc. Ing. Jan Platoš, Ph.D.
vedoucí katedry

prof. Ing. Pavel Brandštetter, CSc.
děkan fakulty

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě 16. dubna 2018 .....

Souhlasím se zveřejněním této diplomové práce dle požadavků čl. 26, odst. 9 Studijního a zkušebního řádu pro studium v magisterských programech VŠB-TU Ostrava.

V Ostravě 16. dubna 2018 .....

Ďakujem vedúcemu mojej diplomovej práce Ing. Svatoplukovi Štolfovi, Ph.D., za jeho odborné vedenie, cenné vedomosti, podnety, pomoc, trpezlivosť a rady, ktoré mi vždy ochotne poskytol pri tvorbe tejto diplomovej práce.

Abstrakt

Diplomová práca sa zaoberá problematikou modelovania biznis procesov podniku a jeho následným overovaním. Naším cieľom bolo využiť techniku modelovania biznis procesov pomocou modelovacieho jazyka UML. Podstatou algoritmu je vykresliť diagram a následne ho overiť podľa predom určených pravidiel ontológie. Pravidlá ontológie tvoria model, ktorý určuje existenciu žiadúcich prvkov a závislosť komponentov v diagrame. Tieto pravidlá overujú korektnosť namodelovaného procesu. Modelované dáta sme spracovali a vďaka nim sme vytvorili štruktúru dát, ktorá slúži ako podklad na analýzu procesu mining.

Klíčová slova: UML, Diagram aktivít, Ontológia, Biznis modelovanie, Proces mining

Abstract

This diploma thesis deals with the problematics of business process modeling and validation. The aim is to use the UML diagramming technique based on an activity diagram. Thanks to the handling drawed data, it was created the activity diagram with requested features. The fundamental logic of the algorithm is to model the diagram and then to verify it according to the predetermined rules of ontology. It was created the system which can store models and diagrams. The diagram has a strong model that validate its correctness. The results of this research provide the data of the business process for process mining analysis

Key Words: UML, Activity diagram, Ontology, Business process modeling, Process mining

Obsah

Seznam použitých zkratek a symbolů

UML	–	Unified Modeling Language
OWL	–	Ontology language
JSON	–	JavaScript Object Notation
XML	–	Extensible Markup Language
JS	–	JavaScript
SQL	–	Štrukturovaný dotazovací jazyk
MS	–	Microsoft
ProM	–	Process Mining tool
HTML	–	Hyper Text Markup Language
BPMN	–	Business Process Model and Notation
IDEF	–	Integration DEFinition language
CSV	–	Comma Separated Values
AD	–	Aktivitový diagram
UTF	–	Unicode Transformation Format
ARIS	–	Automatizovaný rozpočtový informačný systém
ERP	–	Enterprise Resource Planning
WFM	–	Workforce Management
PN	–	Petriho Sieť

Seznam obrázků

Seznam tabulek

Seznam výpisů zdrojového kódu

1 Úvod

Súčasná doba 21. storočia je dynamická, neustále sa meniaca a napredujúca smerom vpred. Postupne sa zvyšujú požiadavky, čím sa vyžaduje reakcia na zmenu, ktorej výsledok by bol adekvátny, korektný, nepôsobiaci zmätok, zníženie kvality či neefektívnosť špecifických modulov. Inak to nie je ani vo svete technológií. Denne prichádzame do kontaktu s požiadavkami na vylepšenie funkcionality produktu, procesu, ale aj na ich vytvorenie od samotných základov podľa vopred stanovených požiadaviek od zákazníka či firmy samotnej.

Podnik, ktorý chce uspieť, musí spĺňať požiadavky a nároky svojich zákazníkov, adekvátne vylepšovať kvalitu svojich produktov a hlavne pružne a pohotovo reagovať na vyžiadané zmeny. Vo sfére biznisu sa nachádza veľmi veľa spoločností, ktoré sa uberajú smerom, ktorý je pre nich špecifický. Keďže dopyt na produkt nie je stabilný a konkurencia je vysoká, je potrebné organizovať v rámci firmy vlastné procesy s čím väčšou efektivitou, požadovanou kvalitou a dokončením v čo najkratšom čase tak, aby spoločný výsledok jednotlivých procesov v rozumnom poradí bol korektný produkt, ktorý je pripravený na predaj zákazníkom, pričom je tu kladený dôraz na skupinovú spoluprácu.

Aby sa mohla spoločnosť rozvíjať a správne napredovať, je žiadúce definovať, vytvárať, upravovať a vizualizovať všetky podnikové procesy tak, aby sa v rámci fungovania podniku dali čo najlepšie aplikovať. Na vizualizovanie a s ňou spojené vytváranie a upravovanie existuje špeciálna technika informačných systémov, ktoré uľahčujú prácu organizácie. Ide o softvérové produkty, ktoré dokážu spĺňať potrebný náhľad procesov vo svojich zahrnutých moduloch a aplikáciách i v ich jednotlivých častiach spojených do skupín. V takomto rozhraní si užívatelia dokážu upravovať svoje procesy a diskutovať v skupinách.

V prvej kapitole bude vysvetlená problematika modelovania podnikových procesov, ktorá v sebe zahŕňa aj explikáciu fundamentálnych pojmov. Bude tu aj rozbor zvolenej metódy modelovania UML. Zoznámime sa s typmi diagramov, ktoré sa využívajú v rámci modelovania podnikových procesov. Na konci tejto kapitoly zhodnotím súčasný stav aplikácie.

V nasledujúcej kapitole popíšem vývojový proces tvoreného softvéru - od špecifikácií požiadaviek cez analýzu, návrh, implementáciu, testovanie a nasadenie. Každá časť bude samostatne popísaná a vysvetlí hlavné kroky danej etapy. Pre lepšie pochopenie riešenia bude využitá vizualizácia pomocou UML diagramov.

V tretej kapitole podrobne popíšem proces modelovania. Vysvetlím hlavnú logiku modelovania, spôsob jej funkcionality. Popísané budú hlavné komponenty diagramu. V nasledujúcej kapitole sa budem koncentrovať na experimenty, ktoré budú prevádzané počas riešenia práce. Zameriam sa na experimenty, ktoré ma dovedú ku konečnému riešeniu.

Ďalšia kapitola bude predstavovať rozbor výsledkov práce, ktoré porovnám so zadaním práce. V poslednej kapitole vyhodnotím prácu, pričom zhodnotím dosiahnuté výsledky práce.

2 Základné informácie

2.1 Zoznámenie sa s problematikou

Podnik, biznis, je komplexný systém, ktorý pozostáva z hierarchickej organizácie určitého odvetvia, ktorý slúži na vyváranie produktu. Proces biznisu, podnikový proces, je po častiach usporiadania množina aktivít a procedúr, ktoré spoločne vedú k dosiahnutiu podnikového alebo strategického cieľa. V procese sa nachádzajú roly a ich vzťahy, ktoré sú v rámci procesu definované. Role sú množina vzájomne sa doplňujúcich vlastností a definujú chovanie, kompetenciu a zodpovednosť osôb alebo skupiny osôb. Niektoré špeciálne aktivity potrebujú na svoje zrealizovanie špecifické prostriedky. Tieto prostriedky sa označujú ako zdroje danej aktivity. Takýmito zdrojmi môžu byť aj ľudské zdroje, konkrétne jednotlivé osoby alebo skupiny osôb, ktoré sú mapované na základe svojich vlastností na role. Hlavnou úlohou podnikového modelovania je vytvoriť abstrakciu daného procesu, na základe ktorej sme schopní dostatočne pochopiť všetky jej aktivity, vzájomné vzťahy medzi nimi a definovanými rolami, ktoré reprezentujú schopnosti účastníkov procesu alebo technických zariadení zahrnutých do daného procesu.

Pod pojmom aktivita rozumieme popis konkrétneho kroku v rámci činnosti procesu. Aktivity sú diferencované na dve skupiny. Prvá skupina aktivít nevyžaduje technickú asistenciu, teda počítačovú podporu. Takýto druh aktivity sa nazýva manuálna aktivita. Naopak, aktivita, ktorá sa neuskutoční bez technickej asistencie, sa pomenúva automatizovaná aktivita (workflow aktivita). Aktivita, ktorá sa prevedie sama bez ľudskej činnosti, sa označuje ako automatická aktivita. Inštancia aktivity charakterizuje konkrétnu činnosť, ktorá sa vykonáva v rámci inštancie procesu. Inštancia procesu znamená konkrétny prípad vykonania procesu. Popisuje, ako sa vykonáva. Inštancia procesu pozostáva z jednotlivých aktivít, začiatku a konca, na ktorom je požadovaný výstup - produkt, služba a podobne.

Podnikový model procesu opísaný takýmto spôsobom je podrobný a komplexný pohľad na podnikový proces. To znamená, že sa tvorí konkrétna abstrakcia. Ukazuje nám, ako sa vykonávajú a ako fungujú funkcie daného procesu. Model pozostáva z toku jednotlivých činností, teda aktivít a zúčastnených rolí, ktoré sú v ňom zahrnuté. Takýto nadhľad nám umožňuje eliminovať nežiadúce chyby a zamerať sa na hlavné aspekty.

Obrázek 1: Podnikový model a jeho komponenty

Efektívny model vedie k detailnému preskúmaniu a diskusii medzi viacerými členmi. Proces musí byť modelovaný pre lepšie pochopenie jeho účelu a v prípade potreby vylepšený podľa požiadavky. Model nerieši všetky problémy, ale slúži ako zjednodušený pohľad štruktúry procesu, ktorý zobrazuje, ako daný proces funguje. Slúži ako základ pre komunikáciu, vylepšenie, inovácie a definuje základné informácie systémových požiadaviek, ktoré sú nevyhnutné na fungovanie procesu. Súhrnom všetkých podnikových procesov v rámci podniku rozumieme podnikový model danej firmy. Popisuje funkcie podniku, ako daný podnik funguje a ako sa realizujú procesy. Jednoducho povedané, popisuje, čo firma robí a ako to robí. [8]

Automatizovaný podnikový proces sa označuje pojmom Workflow (tok prác). Zásadný rozdiel medzi workflowom a podnikovým procesom je ten, že workflow je automatizovaný. Takýto proces spravovaným softvérom (ERP alebo WFM systém).

V ideálnom prípade podnikový model pozostáva z jedného diagramu, ktorý zachytáva a obsahuje všetky dôležité aspekty podniku. Dosiahnuť takýto stav je nemožné, pretože podnik je komplexný a má mnoho aspektov, ktoré nie je možné zachytiť jedným diagramom, ktorý by obsahoval všetky potrebné informácie na plné zobrazenie. Práve z tohto dôvodu je podnikový model štruktúrovaný nasledovnými komponentami:

- **Pohľady (Views):** Podnikový model je ilustrovaný množstvom pohľadov, ktoré sú samostatné, čiže obsahujú jeden alebo niekoľko špecifických aspektov podniku. Pohľad je konkrétna abstrakcia určitého aspektu, ktorý obsahuje potrebné informácie. Viacnásobné pohľady je potrebné rozdeľovať tak, aby pri separovaní nedošlo k strate žiadnych dôležitých informácií. Je nutné minimalizovať viacnásobné pohľady, aby sa zaistila jednoduchosť, korektnosť a zminimalizovala závislosť.

- **Diagramy:** Každý pohľad pozostáva z určitého rozumného množstva diagramov, pričom každý z nich zachytáva špecifickú štruktúru podniku alebo podnikovú situáciu. Niekoľko diagramov zobrazuje rovnaký pohľad. Je to z toho dôvodu, že každý typ diagramu má

vlastnosti a účel na zachytávanie iných vlastností aspektu podnikového modelu. Pomocou nich možno ukázať štruktúru, chovanie, vzťahy, spoluprácu a podobne. Obsahujú i objekty, procesy, pravidlá, ciele, vízie, priebeh funkcie a mnoho ďalších.

- **Objekty a procesy:** Obsah je štrukturovaný v diagramoch vďaka použitiu rozličných objektov a procesov. Objektami rozumieme „veci“ v podniku. Môžu byť fyzické (ako napríklad: osoba, ľudia, stroje, produkty) alebo abstraktné (sem zahŕňame inštrukcie, chovanie funkcií, služby atď.). Objekty môžu obsahovať aj ostatné objekty o ostatných objektoch. Procesy sú funkcie v podniku, ktoré vytvárajú, menia, spotrebujú alebo užívajú objekty k vytváraniu ďalších objektov.

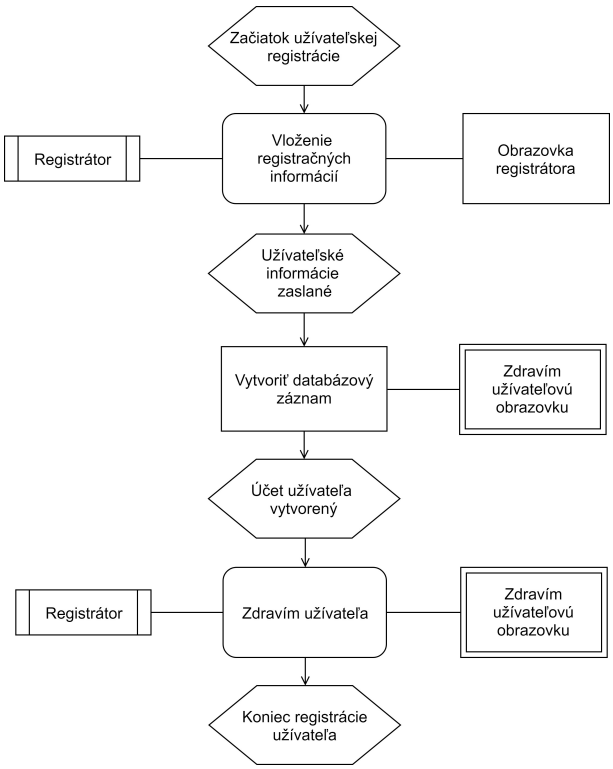
Tieto jednotlivé komponenty tvoria systém procesného inžinierstva, ktorý ma jasne definované všetky svoje pojmy i vzťahy, ktoré spolu ako celok slúžia na reprezentáciu daného procesu. Konkrétnou ontológiou rozumieme zoskupenie jednotlivých aktivít, rolí, zdrojov a podobne, ktoré spoločne tvoria celok. Sú v nej popísane aj vzťahy a závislosti. [2] Existujú dve základné formy modelovania abstrakcií:

1. **Neformálne (semi-formálne) metódy modelovania podnikových procesov:** Pri týchto prístupoch je zachytený základný prístup modelovania. Konkrétne ide o funkčné modelovanie, štrukturálne modelovanie a modelovanie chovania. K týmto druhom modelovania sa využívajú nasledovné metódy:

   - **Metóda IDEF (Integrated Definition):** Daná metóda obsahuje jazyk, pomocou ktorého sa tvorí štruktúrovaná grafická reprezentácia systému. Má vlastný špecifický jazyk, ktorý má pevne danú syntax a sémantiku svojho jazyka. Výsledkom je model, ktorý je konzistentný s popisom systémových funkcií, charakteristickými vzťahmi medzi nimi a príslušnými informáciami. Konkrétne sa skladá z metód IDEF0 (zahŕňa základný popis a poradie jednotlivých funkcií), IDEF1 (obsahuje informačný model, v ktorom je obsiahnutá štruktúra a sémantika informácií), IDEF2 (opisuje dynamické chovanie systému).
   - **Metóda UML (Unified Modeling Language): Metóda UML (Unified Modeling Language):** Metóda UML má vlastný charakteristický jazyk UML, ktorého hlavnou úlohou je zjednotenie prístupu pri vytváraní špecifikácií v rámci softvérového procesu. Má široké využitie pri vytváraní grafickej dokumentácie procesov, modulov i systémov. Táto diplomová práca je zameraná na tento konkrétny prístup modelovania podnikových procesov. Metóda UML je bližšie špecifikovaná v nasledujúcej podkapitole. [1]
   - **Metóda EPC (Event-Driven Process Chains):** Uvedená metóda pozostáva z reťazenia aktivít a metód do takej konečnej postupnosti, ktorú daná postupnosť vykoná a prevedie požadovaný cieľ. Každá udalosť je definovaná vstupnou podmienkou (precondition), ktorá sa podieľa na realizovaní aktivity. Ukončenie danej aktivity

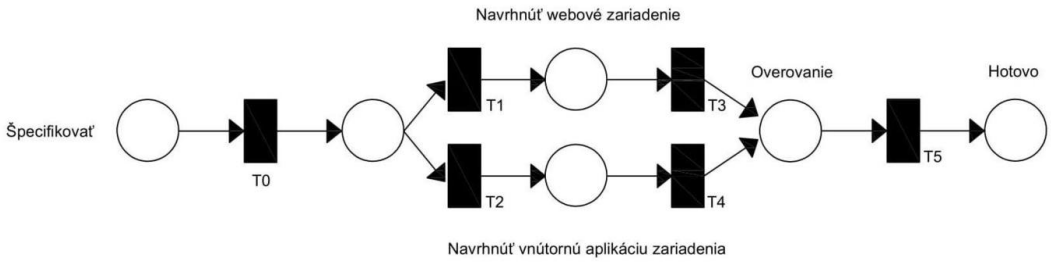
je definované výstupnou podmienkou (postcondition), ktorá slúži ako základ alebo nadväznosť na nasledujúcu aktivitu. Jednoducho povedané, aktivita je obmedzená vstupnou a výstupnou podmienkou danej aktivity. Je súčasťou rozvinutých systémov, ako sú napríklad systém SAP, ARIS. [13]

2. **Formálne metódy modelovania podnikových procesov:** Metóda modelovania Petriho sietí, ktoré zachytávajú podrobnejšie a jednoznačne špecifikácie procesov, pričom zachytávajú verifikáciu aj analýzu daného procesu. [14]



Obrázek 2: Ukážka EPC diagramu

Na Obrázku 2 vidíme diagram registrácie užívateľa vytvorený metódou EPC, kde je zreteľne zobrazená vstupná a výstupná podmienka jednotlivých aktivít. Je tu očividná postupnosť i tok aktivít a zdrojov.



Obrázek 3: Petriho sieť

Teraz sme si vymenovali a stručne vysvetlili jednotlivé metódy modelovania podnikových procesov. Každá z týchto metód používa vlastný jazyk, rôzne prístupy a abstrakcie. Na druhej strane je ale badateľné, že vo všetkých vystupovali zdroje, objekty, aktivity, roly, vzťahy a podobne. Môžeme povedať, že všetky obsahujú rovnakú špecifikáciu, majú teda rovnaký meta-model modelovania procesov. Meta-model je model, ktorý definuje jazyk určený na vytvorenie modelu. Ide o model, ktorého úlohou je popísať model.
Ku koncu tejto kapitoly si ešte zhrňme pre nás najdôležitejší pojem, aby sme správne chápali, čo je predmetom tejto práce.

Podnikový proces je aktívna časť podniku. Opisuje funkcie podniku a zapája zdroje, ktoré sú použité, transformované a produkované. Podnikový proces je abstrakcia, ktorá ukazuje spoluprácu medzi zdrojmi a transformáciu zdrojov v podniku. Zdôrazňuje, ako je práca prevádzaná, nad tým, ako je opísaný produkt alebo služba ako výsledok procesu. Viac formálny opis podnikového procesu je nasledovný:

Podnikový proces je kolekcia aktivít, ktoré berú jeden a viac druhov vstupov a vytvárajú výstup pre zákazníka. Podnikový proces má cieľ a je simulovaný udalosťami v externom svete alebo v ďalšom procese.

Pre zosumarizovanie definícií, podnikový proces:

- Má cieľ.
- Má špecifický vstup.
- Má špecifický výstup.
- Používa zdroje.
- Má množstvo aktivít, ktoré sú prevádzané v určitom poradí v závislosti od podmienok a udalostí, ktoré nastávajú počas prevádzania procesu. Aktivity v rámci procesu môžu byť videné ako podprocesy.
- Zachytávaja viac ako jeden celok. Je orientovaný prevažne horizontálne než vertikálne v závislosti od tradičnej organizácie podniku.

- Vytvára hodnoty určitému druhu zákazníkov. Zákazník vo vzťahu k podniku môže byť interný alebo externý.

2.2 Jayzk UML

Unified Modeling Language je súbor grafických značení a notácií, ktoré sa používajú najmä pri vývoji softvéru. Svoje využitie má aj v rámci vývoja produktu rozličných produktov alebo služieb organizácie. Je štandardom analýzy a návrhu. Slúži ako uľahčujúci nástroj na návrh a vývoj systému, služby či produktu. Je objektovo orientovaný a vytvára objektovo orientované modely, čím dostatočne zachytáva abstrakciu špecifickej reality.

Unified Modeling Language pozostáva z mnoho rozličných typov diagramov a každý z nich zobrazuje špecifické statické alebo dynamické aspekty systému. Má silnú konštrukciu pre modelovanie.

- Ako využívajú rôzni aktéri daný proces?
- Aké aktivity sú obsiahnuté v danom procese? Ktoré sú špecifické pre konkrétny druh aktéra?
- Čo je primárnym, ultimátnym cieľom ich práce?
- Čím sú jednotliví aktéri špecifickí pri prevádzaní procesu?
- Aké sú pravidlá jednotlivých aktivít a aká je ich štruktúra?
- Existujú nejaké vylepšenia pre proces, aby bol prevádzaný efektívnejšie?

Pomocou modelovania podnikových procesov a správneho chápania procesu dokážeme modelovať podnikový proces tak, aby vyhovoval požiadavkám. Môže simulovať už fungujúci proces, ktorý možno analyzovať a prípadne upraviť tak, aby stále spĺňal požadovaný cieľ. Taktiež je uskutočniteľné modelovanie nových procesov a ich postupné vylepšovanie.[7]

Pomocou objektovo orientovaného jazyka UML dokážeme vytvárať podobné koncepty na základe reality, čím tvoríme podnikový procesný model, ktorý dokáže zachytiť potrebné funkcie s potrebnými vlastnosťami. Týmto postupom sa dostávame k naplneniu cieľa. Jednotlivé aktivity, zdroje, roly, závislosti dokážeme zachytiť a mapovať do objektov, čím vytvárame statický alebo dynamický model. Využívame osvedčené techniky objektového modelovania a štandardnú notáciu, ktorá je žiadúca pre jednotné značenie sémantiky. Tento modelovací jazyk má vlastnú notáciu - symboly použité v modeloch - a určuje pravidlá riadenia jazyka. Pravidlá sú syntaktické, sémantické a pragmatické. [1]

Syntaktické pravidlá diktujú, ako majú symboly vypadať a ako majú byť kombinované. Syntax môže byť zrovnateľná do slov v prirodzenom jazyku, pretože je dôležité vedieť, ako im hovoriť a ako kombinovať ich klauzuly. Sémantické pravidlá poskytujú deskripciu toho, čo každý symbol znamená a ako má byť interpretovaný sám sebou alebo konceptom symbolov. Tieto

pravidlá sú porovnateľné s definíciami slov v prirodzenom jazyku. Podľa slov vieme určiť, čo má daný prvok znamenať. Pragmatické pravidlá vysvetľujú, ako použiť modelovací jazyk UML. Je určovaný smernicami v prirodzenom jazyku. Definuje zámery, usmernenia symbolov, vďaka ktorým je účel modelu dosiahnutý, zrozumiteľný a porozumiteľný.

Ak chceme vyvinúť návrh systému od koncepcie k podrobnému objetovo orientovanému návrhu, musíme uskutočniť niekoľko činností:

1. Pochopiť a definovať kontext a externé interakcie systému.
2. Navrhnúť systémovú architektúru.
3. Identifikovať základné objekty v systéme.
4. Vyvinúť modely návrhu.
5. Špecifikovať rozhranie.

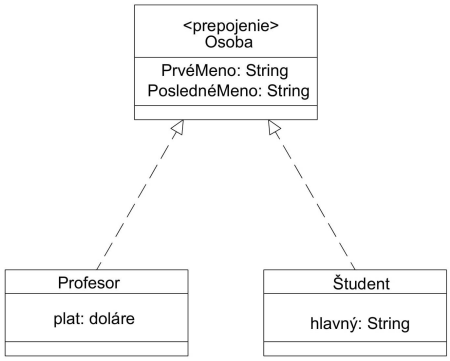
Podobne ako iné kreatívne aktivity nemá ani návrh pevne daný postup. Pri vývoji návrhu sa vychádza z nápadov, navrhujú sa riešenia a tieto riešenia sa doladzujú na základe novozískaných informácií. Keď sa vyskytnú problémy, je nevyhnutné vrátiť sa späť a skúsiť iné riešenie. Pri jednotlivých možnostiach sa niekedy podrobne skúma, či budú fungovať. Inokedy sa dá podrobnosti ignorovať až do neskoršej fázy procesu. Z tohto dôvodu tento proces zámerne neilustrujeme jednoduchým diagramom, ktorý by naznačoval, že si návrh môžeme predstaviť ako jednoduchú sekvenciu aktivít. Všetky vyššie uvedené aktivity sú v praxi poprepájané a navzájom sa ovplyvňujú.

UML rozdeľujeme na tri základné prístupy, pomocou ktorých modelujeme procesy:

1. **Funkčný prístup:** Zaoberá sa štruktúrovaním funkcií na ich vstupy a charakteristické výstupy. Je zameraný na dynamický aspekt.
2. **Prístup chovania:** Je zacielený na riadiaci aspekt vykonávania procesu po sebe nasledujúcich udalostí a podmienok, za ktorých sú jednotlivé aktivity uskutočnené.
3. **Štrukturálny prístup:** Jeho úlohou je zachytiť entity a zdroje zainteresované v procese. Obsahuje informácie o zdrojoch a entitách, ich atribúty, činnosti a závislosti medzi nimi. [3]

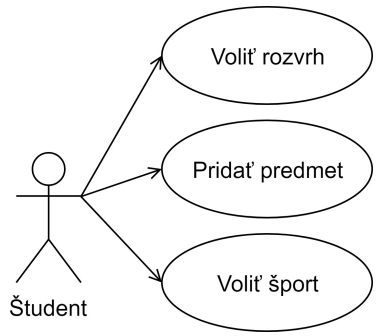
Teraz stručne uvediem príklady diagramov modelovania procesov pomocou UML:

- **Triedny diagram:** Popisuje štruktúru systému. Štruktúra pozostáva z tried a vzťahov. Triedy majú reprezentačnú funkciu, štruktúrujú informácie, produkty, dokumenty alebo organizáciu.



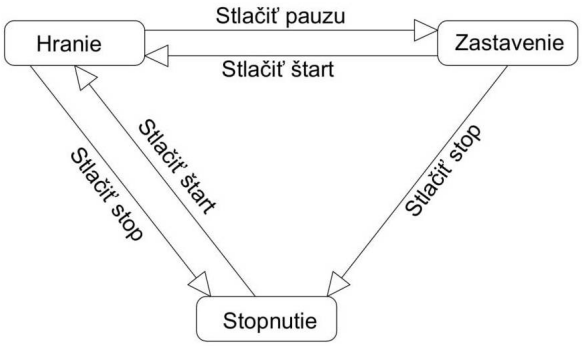
Obrázek 4: Ukážka triedneho diagramu pre osoby popisujúce učiteľa a študenta hry

- **Objektový diagram:** Vyjadruje možné kombinácie objektov špecifického triedneho diagramu. Obvykle je používaný na ilustrovanie triedneho diagramu.
- **Stavový diagram:** Vyjadruje možné stavy systému alebo určitého objektu. Môže byť zobrazený aj stavovým automatom.



Obrázek 5: Diagram prípadu užitia pre študenta

- **Diagram aktivít:** Popisuje aktivity a akcie prebiehajúce v systéme.
- **Sekvenčný diagram:** Zachytáva jednu alebo niekoľko sekvencií správ posielaných naprieč skupinou objektov.
- **Diagram spolupráce:** Popisuje kompletnú kooperáciu medzi skupinou objektov.
- **Diagram prípadu využitia:** Ilustruje vzťahy medzi prípadmi využitia. Každý prípad použitia typicky definovaný ako čistý text opisuje časť celkovej funkcionality systému.



Obrázek 6: Ukážka stavového diagramu pre zastavenie a spustenie hry

- **Diagram komponent:** Ide o špeciálny typ triedneho diagramu, ktorý sa používa na opis hardvéru, ktorý je vnútri softvérového systému.

2.3 Súčasný stav aplikácie

V tejto podkapitole je opísaný stav aplikácie na modelovanie biznis procesov v dobe, keď sa na tejto diplomovej práci začalo postupne pracovať. Stav, v ktorom mi bola predaná aplikácia, bol v štádiu výskumu, čo znamená, že táto verzia nemusí splňovať konkrétne požiadavky, funkcionalitu a riešenie, ktoré boli pre dokončenie práce nevyhnutné. Výsledky, ktoré obsahuje táto verzia, sú bližšie opísané v nasledujúcich riadkoch. Taktiež je tu vylíčená i stručná deskripcia jej nadväznosti na ďalší vývoj.

Výskum pozostával zo štúdia literatúry a zdrojov. Na základe preskúmania zdrojov bol v minulosti navrhnutý proces vývoja počiatočnej verzie aplikácie, na ktorú sa následnými integráciami nadväzovalo tak, aby daný proces postupne dopĺňal funkcionalitu. Predošlý autor skúmal problematiku a snažil sa navrhnúť vhodne štruktúrovaný softvér, ktorého hlavná úloha bola modelácia biznis procesov a následné overenie ich korektnosti pomocou preddefinovaných pravidiel a vytvoreného meta-modelu pre samotné modelovanie a jeho nadväznosť na proces mining.

V dokumentácii projektu boli vizualizované ciele v dobe prebiehajúceho vývoja, ale aj plány do budúcnosti, ktoré obsahovali vizuálne návrhy, čo by mala aplikácia spĺňať v rámci jej funkcionality. Postupne bol započatý softvérový vývoj.

Aplikácia obsahovala spoluprácu s frameworkom na vygenerovanie layoutu aplikácie, kde sa nachádzali hlavné komponenty. Takisto obsahovala databázu a jednoduché, základné prepojenie. Konkrétne išlo o webovú aplikáciu v prostredí .Net frameworku písanú v objektovo orientovanom jazyku C Sharp. Využívaná bola databáza Microsoft Sql Server 2012, s ktorou aplikácia jednoducho komunikovala. Ďalej obsahovala prvky HTML, CSS a JavaScript.

Na spomínanú prácu sme nadväzovali so spolužiakom Bc. Miroslavom Babralom, s ktorým sme diskutovali o vtedajšom stave aplikácie, možnom vylepšení, integrovaní a samotnej implementácii, pretože moja práca nadväzovala na jeho diplomový projekt – Proces Mining. Na túto prácu sme nadväzovali spoločne s tým rozdielom, že moja časť je primárne určená na mode-

lovanie a overovanie modelovania procesov. Miroslav nadväzoval na výsledky mojej práce, na základe ktorých sa venoval problematike proces miningu. Naša kooperácia je opísaná v kapitole implementácií, kde je jasne definované, ktorú časť pri našej spolupráci tvoril Miroslav a ktorú som vytváral ja.

Pri hlbšom skúmaní problematiky som pochopil, že výskum projektu prebiehal v poriadku, ale návrh, architektúra i implementácia sa uberali pre moju prácu nesprávnym smerom, čo mi do značnej miery skomplikovalo riešenie. Dizajn aplikácie pre modelovanie biznis procesov bol rozložený nekorektne, neobsahoval potrebnú logiku, nedokázal udržať konzistenciu modelovaných dát. Z daného dôvodu prvý krok predstavoval navrhnutie správneho dizajnu aplikácie s potrebnými modelovacími funkciami.

Skôr, ako som sa pustil do úpravy aplikácie, pokračoval som v hlbšom skúmaní jednotlivých komponentov softvéru. Prišiel som na to, že databáza už viac nie je dostupná a nebola vhodne navrhnutá pre moju prácu.

Po dokončení skúmania štruktúry, architektúry a implementácie samotného programu, ale aj databázového systému, sme diskutovali o vtedajších problémoch a riešeniach spolu s vedúcim tejto diplomovej práce. Počas mítingov, ktoré sme organizovali, sme sa na základe výsledku súčasného výskumu môjho štúdia danej problematiky modelovania biznis procesorov, Miroslavovho štúdia proces miningu a odborných rád vedúceho diplomového projektu spoločne rozhodli, že začnem aplikáciu budovať od samotných základov. Postupne sme určovali a zvyšovali ciele jednotlivých procesov tak, aby boli schopné pružne reagovať na zmenu.

Navrhol som vhodnú architektúru, spravil som podrobný softvérový vývoj a postupne som začal tvoriť aplikáciu na modelovanie podnikových procesov a s nimi spojeným overovaním korektnosti. Jednotlivé kroky sú podrobne popísané v nasledujúcej kapitole, ktorá sa zaoberá softvérovým vývojom.

3 Softvérový vývoj aplikácie

Pri vývoji aplikácie som postupoval štandardným vývojovým procesom, ktorý pozostával zo špecifikácie požiadaviek, analýzy problému, návrhu riešenia problému, implementácie, testovania aplikácie a zavedenia do prevádzky.

3.1 Špecifikácia požiadaviek

Špecifikácia požiadaviek bola prevádzaná podľa IEEE štandardu. Vytvoril som dokument špecifikácie požiadaviek, ktorý slúži na zachytávanie funkčných a nefunkčných požiadaviek na systém.

3.1.1 Úvod

Účel: Táto špecifikácia požiadaviek na softvér opisuje užívateľské, funkčné a parametrické požiadavky softvéru, ktorého účel je zameraný na modelovanie podnikových modelov. Je určený pre podniky, organizácie a spoločnosti na zachytávanie svojich procesov. Tento podnikový model je definovaný svojím meta-modelom, pomocou ktorého budeme schopný zobrazovať jeho účel, funkcie, zdroje a podobne. Softvér ponesie názov UML Business Modeler. UML Business Modeler slúži pre podniky, ktoré chcú vizualizovať svoje procesy. Výsledkom modelovanej práce bude korektný diagram aktivít, ktorý bude spĺňať preddefinované OWL pravidlá, ktoré si užívateľ vytvorí pomocou nástroja Protégé. Výsledný korektný diagram bude slúžiť ako podklad pre vyhodnocovanie podnikových procesov a taktiež sa bude dať exportovať vo formáte JSON. Prvá verzia softvéru bude obsahovať a validovať aktivitový diagram, ktorého vlastnosti sú definované podľa jazyka UML.

Rozsah projektu a funkcie systému: Systém pre modelovanie podnikových procesov, UML Business Modeler, bude poskytovať funkcie, ktoré budú slúžiť na modelovanie daného procesu. Taktiež bude schopný načítať pravidlá otológie vo formáte OWL, vyhodnotiť a zvalidovať namodelovaný diagram aktivít. Po modelovaní vyhodnotí korektnosť modelovaného diagramu. Diagramy a pravidlá, ktoré si vytvorí užívateľ, bude spôsobilý ukladať. Úlohou tejto verzie je vytvoriť pevný a korektný základ na dopĺňanie ďalších UML diagramov a s nimi potrebnźch validácií.

Slovník pojmov:

- **OWL:** ontológia, ktorá definuje pravidlá diagramu
- **UML:** modelovací jazyk
- **Diagram:** grafická reprezentácia toku aktivít, poprepájaných vzťahov a využitých zdrojov
- **Validita diagramu:** podľa pravidiel ontológie vyhodnotí, či tieto pravidlá diagram obsahuje a korektne dodržuje

- **JSON:** spôsob zápisu dát nezávislý na počítačovej platforme, výsledok je vždy reťazec, má svoju hierarchiu
- **Protégé:** externý program, ktorý slúži ako editor pravidiel otológie
- **Modeler:** nástroj na modelovanie
- **Diagram aktivít:** diagram na modelovanie aktivít procesov

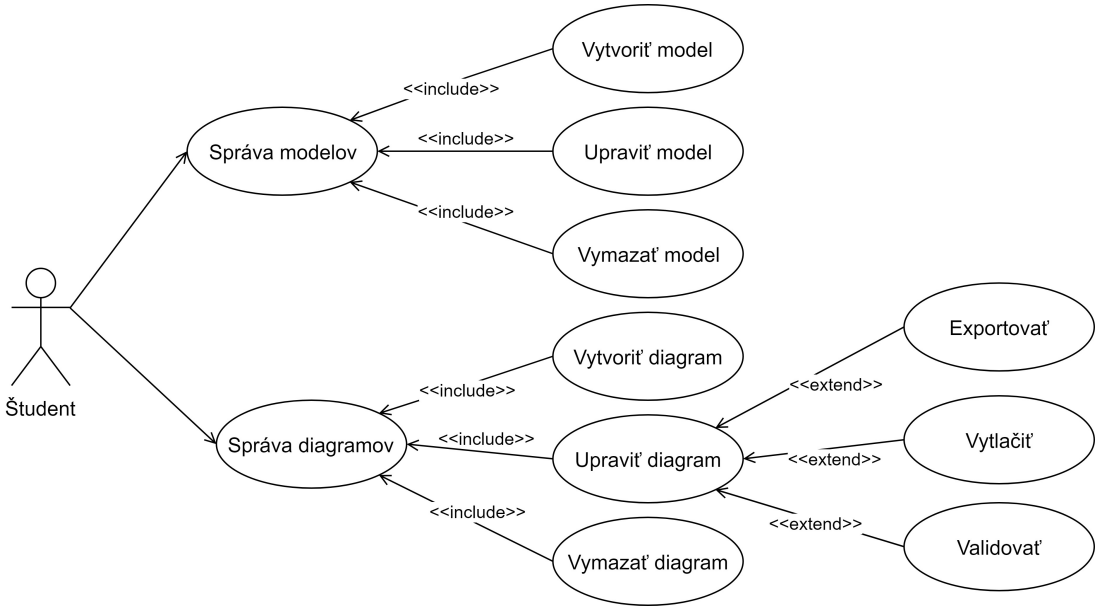
Odkazy:

- Kniha Softvérové inžinierstvo od autora Iana Sommervilla
- Kniha Business Modeling with Uml od autorov Hansa-Erika Erikssona a Magnusa Penkra
- Kniha Software Engineering: A Practitioner's Approach od autora Rogera Pressmana
- Kniha Workflow Modeling: Tool for Process Improvement and Aplication Development od autorov Aleca Sharpa a Patricka McDermotta

3.1.2 Všeobecný popis

Kontext systému: UML Bussines Modeler predstavuje modelovací nástroj UML diagramov, ktorý je závislý na svojom meta-modeli. So softvérom pracuje zákazník, ktorý pomocou neho modeluje podnikové procesy v rámci svojej organizácie. Systém komunikuje s uloženými ontologickými pravidlami, ktoré si ukladá užívateľ do systému. Komunikuje s databázovým systémom. Po uložení ostanú pre neho dostupné na editáciu, vymazanie a hlavne na modelovanie diagramu. Diagram sa exportuje vo formáte JSON, ktorý sa ďalej využije na proces mining pre program ProM.

Prehľad funkcií: Užívateľ softvéru bude schopný vytvárať vlastné diagramy. Bude mať prístup k svojim modelom a diagramom, ktoré bude môcť spravovať. Funkcie softvéru sú popísané nasledovným diagramom prípadu užitia.



Obrázek 7: Prípady využitia aplikácie

Softvérové rozhranie: UML Business Modeler bude vytvorený ako webová aplikácia v technológii ASP.NET Core Framework, MySQL.

Užívateľské rozhranie: Softvér bude korektne pracovať na požadovaných internetových prehliadačoch Opera 51.0, Google Chrome 65.0, Microsoft Edge 41.0, Mozila FireFox 59.0. Rozhranie bude mať intuitívnu logiku a jednoduché ovládanie. Bude zladené farebnými schémami tak, aby pôsobilo prirodzene a s minimom farebných kombinácií.

Triedy používateľov a ich vlastnosti:

- **Zákazník:** Zákazníkom je ľubovoľná právnická alebo fyzická osoba, ktorá má poverenie od organizácie modelovať podnikové procesy spoločnosti. Zákazníkom rozumieme aj študenta, ktorý má záujem o modelovanie procesov.

Obmädzujúce pravidlá:

1. Keďže modelovanie a následné validovanie diagramu je závislé na preddefinovaných pravidlách pomocou externého programu Protégé, je obmedzujúcou podmienkou mať predom vytvorené pravidlá ontológie vo formáte súboru typu owl, ktorý bude schopný ukladať ich do systému a pracovať s nimi.

2. Program bude schopný modelovať meta-model, taktiež bude obsahovať práve jeden štartovací uzol, koncový uzol, aktivity, rozhodovacie a zlučovacie uzoly a prechody medzi uzlami.

3.1.3 Špecifikácia požiadaviek

Use Case 1: Správa modulov diagramu

Popis: Užívateľ systému si vyberie z hlavného menu možnosť spravovania diagramov.

Vstupné podmienky: Užívateľ sa nachádza v systéme.
Výstupné podmienky: Podľa zvolenej funkcie - vytvoriť, upraviť, vymazať.
Bežná cesta:

1. Užívateľ zadá požiadavku na zobrazenie všetkých diagramov.
2. Systém zobrazí zoznam všetkých diagramov.
3. Užívateľ zvolí požiadavku na konkrétny diagram alebo na vytvorenie nového diagramu.
4. (a) Systém prijíme požiadavku na úpravu a zobrazí editačnú stránku.
    i. Užívateľ modeluje, upravuje diagram a zadá požiadavku na uloženie, overenie alebo export diagramu podľa definovaných pravidiel.
    ii. Systém prijíme požiadavku a vyhodnotí požiadavku.
    iii. Systém zobrazí výstup požiadavky.
   (b) Systém prijíme požiadavku a zobrazí formulár na tvorbu nového diagramu.
    i. Užívateľ zobrazí stránku na tvorbu nového diagramu.
    ii. Užívateľ vytvorí podľa ponuky nový diagram a uloží ho.
    iii. Systém uloží diagram do systému a zobrazí hlavné menu.
   (c) Systém prijíme požiadavku na vymazanie diagramu a žiada potvrdenie od užívateľa.
    i. Užívateľ potvrdí požiadavku.
    ii. Systém vymaže diagram zo systému a zobrazí hlavné menu.
5. Prípad použitia končí.

Use Case 2: Správa modulov modelov
Popis: Tento prípad využitia prebieha takmer rovnako ako správa modulov diagramov s tým rozdielom, že obsahuje iba základné údaje. Neobsahuje žiadne validovanie podľa OWL pravidiel, no na druhej strane je potrebné podotknúť, že tieto prdavidlá definuje.

Funkčné požiadavky:

1. Každý model je jedinečný, všetky jeho úpravy sa viažu iba na tento konkrétny model. Je jedinečne identifikovaný podľa svojho identifikačného čísla.
2. Každý diagram je jedinečný, všetky jeho úpravy sa viažu iba na tento konkrétny diagram. Tiež je jedinečne identifikovaný podľa svojho identifikačného čísla.
3. Každý užívateľ má jedinečné meno a identifikačné číslo.
4. Každý diagram má pevne naviazaný práve jeden model pravidiel a práve jedného užívateľa.
5. Každý model je prístupný na tvorbu mnoho diagramov.
6. Výsledok validácie je viditeľný užívateľovi.

3.1.4 Štúdium prevedenia

Štúdium prevedenia je krátke a cielené štúdium, ktoré je vhodné prevádzať na začiatku procesu inžinierstva požiadaviek. Má odpovedať na tri kľúčové otázky:

1. Prispieva systém k dosiahnutiu hlavných cieľov organizácie?
2. Je možné systém implementovať v plánovanom čase a s vymedzeným rozpočtom pomocou aktuálnej technológie?
3. Je možné systém integrovať s inými používanými systémami?

Ak je odpoveď na niektorú z týchto otázok záporná, pravdepodobne nie je vhodné na projekte pokračovať.

Odpovede na jednotlivé otázky:

1. Áno. Takýto systém rieši problém tejto práce modelovania podnikových procesov. Je navrhnutý korektne na základne správne definovaného meta-modelu a za pomoci vytvorených i overených pravidiel ontológie pomocou externého programu Protégé. Výsledný diagram v tvare formátu JSON slúži ako podklad na spracovanie procesu miningu.
2. Áno. Tento systém možno prevádzať na infraštruktúru operačného systému Linux aj Microsoft Windows, pretože je vyvinutý v najmodernejšej technológii .Net Core frameworku. Hlavnou úlohou tohto výskumu je implementovať predovšetkým modelovanie podnikových procesov na základne zvoleného meta-modelu a overiť správnosť namodelovaného modelu podnikového procesu pomocou pravidiel ontológie.
3. Keďže začíname s prvou verziou softvéru, jsoftvéru, tato otázka je považovaná za splniteľnú.

Výsledkom štúdia prevedenia sme dostali záver, že je možné a správne v softvérovom vývoji pokračovať tak, aby spĺňal nároky naň kladené. Prechádzame na ďalšiu fázu - analýzu problému.

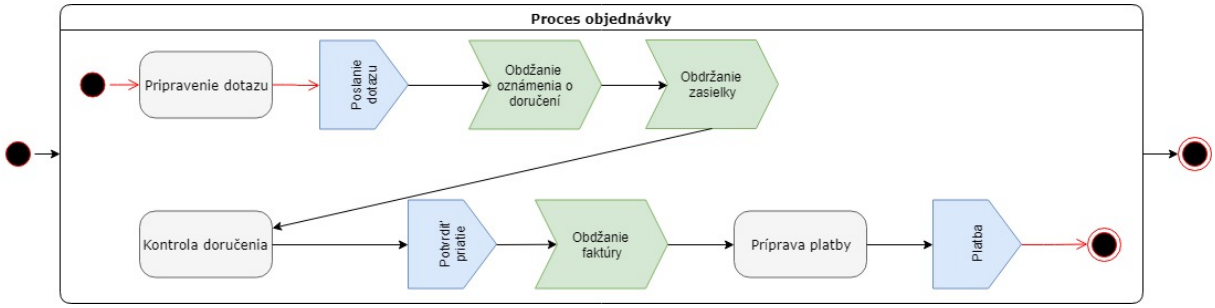
3.2 Analýza problému

Analýza pozostáva z tvorby konceptuálneho modelu. Máme špecifikované požiadavky. Vychádzame z výskumu. Ako základ na modelovanie procesov pomocou diagramu aktivít je využívaný modelovcí jazyk UML. Týmto diagramom dokážeme zachytiť funkcie procesu, ako a za akých podmienok prebieha, v akom poradí a s akými zdrojmi.

Diagram aktivít sa používa na skúmanie a opisovanie workflowových akcií prevádzaných v akciách v triedach. Je podobný tradičnému vývojovému diagramu. Diagram aktivít sa používa na opísanie podnikových procesov, pracovného postupu určitého kontextu organizácie. Dokáže to vďaka ilustrovaniu postupu sekvenciou aktivít. AAktivita je jednoduchá - ako napríklad vytvoriť

objednávku v objednávacom systéme - alebo viac komplexná - ako napríklad kontrola produkcie a samotný vývoj. [1]

Je nutné dobre pochopiť princípy diagramu aktivít, preto v tejto analýze predvedieme modelovanie podnikového procesu. Rozoberieme pre nás dôležité prvky tohto diagramu, ktoré budeme využívať.



Obrázek 8: Diagram aktivít opisujúci proces objednávky

Na prechadzajúcom obrázku je ilustrovaný diagram aktivít pre objednávkový proces. Objednávka začína s aktivitou pripravenia dotazu objednávky a potom posiela samotný dotaz. Po tom, čo je zaslaný dotaz, objednávkový proces čaká na oznámenie o doručení. Keď je zásielka obdržaná, je skontrolovaná a prijatá. Toto prijatie je zaslané dodávateľovi. Faktúra je obdržaná objednávkovým procesom a platba je pripravená a zaslaná. Nie sú tu žiadne špecifikované udalosti prechodov, pretože diagram má automatické prechody k ďalšej aktivite.

Diagram aktivít má meta-model, ktorý pozostáva z aktivít, počiatočného a koncového uzla, rozhodovacieho uzla, prechodov, zdrojov, zlučovacieho uzla a rolí. Pre náš meta-model sme si zvolili prvky aktivít, rozhodovacieho, zlučovacieho uzla, počiatočného, koncového uzla a prieheh pocesu popisujúci šípkami.

Teraz budeme analyzovať prvky nášho meta-modelu a popíšeme jednotlivé komponenty tak, ako budú reprezentované v našom diagrame:

- **Aktivitový uzol:** Uzol zobrazuje aktivitu, ktorá je prevádzaná v rámci definovaného podnikového procesu. Aktivity sú prevádzané a prechod z aktivity značí ukončenie aktivity a prechod k ďalšej. Po prevádzaní aktivity sa automaticky posunie k ďalšej aktivite. Je to z toho dôvodu, že prechody sú prevádzané automaticky, preto nie sú potrebné žiadne udalosti. Tento uzol môže byť rozdelený do podaktivít. Podaktivity, ktoré nemôžu byť rozdelené na ďalšie podaktivity, sa nazývajú akcie (automatické aktivity). Aktivitový symbol je charakterizovaný špeciálnym symbolom. Je označený obdĺžnikom so zaoblenými hranami.

- **Rozdeľovací a spojovací uzol:** Rozdeľovací uzol slúži na oddelenie priebehu vykonávania procesu. Charakterizujeme ním základné podmienky vykonávania funkcií procesu. Prichádza do neho vstupná podmienka, ktorá môže nastať v rámci vykonávania aktivity.

Spojovací uzol súži na spojenie toku aktivity, ku ktorej smeruje. Neslúži na paralelné spracovanie, ale iba na zoskupenie vstupných procesov bez toho, aby musel čakať na všetky vstupy. Stará sa o synchronizáciu. Oba uzly sú charakterizované diamantom s tým rozdielom, že rozhodovací uzol má v sebe text, ktorý charakterizuje podmienku. Zlučovací uzol text neobsahuje.

- **Štartovací uzol:** Indikuje stav, z ktorého začína vykonávanie procesu. Je zobrazovaný kruhom vyplneným farbou, väčšinou bledých odtieňov. V našom diagrame bude obsahovať v sebe text "Start".
- **Koncový uzol:** Indikuje stav, v ktorom končí vykonávanie procesu. Zobrazujeme ho dvojitou kružnicou. Kružnica bližšie k stredu je plnej farby, väčšinou čiernou, a druhá kružnica určuje obrys.V našom diagrame bude obsahovať text "End".
- **Tok procesu:** Tok procesu je spojenie uzlov diagramu v rámci vykonávania podnikového procesu. Obsahuje konečnú postupnosť krokov, ktoré sa vykonávajú. Je charakterizovaný šípkami, ktoré sa skladajú zo zdroja a nasledovníka. Prvotný tok začína od štartovacieho uzla a končí v konečnom elemente diagramu.

Pri vývoji systému som využíval inkrementálny vývoj, pretože funguje na tom princípe, že sa najskôr tvorí počiatočná implementácia, ktorá sa predkladá užívateľom ku komentovaní a vyvíja sa niekoľko verzií, pokiaľ nevznikne požadovaný systém. Procesy špecifikácie, vývoja a validovania nie sú samostatné, ale vzájomne sa prelínajú. Medzi jednotlivými aktivitami pritom funguje rýchla spätná väzba. Na začiatku procesu sme hrubo definovali požiadavky na systém. Projekt sa rozdelí na prírastkové verzie a každá táto prírastková verzia sa vyvíja paralelne alebo samostatne. Za kľúčový faktor tohto vývoja sa považuje možnosť overenia funkcií zákazníkom po každom novom inkremente systému. Ďalším hlavným faktorom je správne rozdelenie vývoja do samostatných inkrementov, samostatných prírastov systému. Nevýhodou môže byť vytvorenie architektúry ešte vo chvíľach, keď nie sú zrejmé všetky definované požiadavky na systém.

3.2.1 Analýza Modelovania

Niektoré z nasledujúcich knižníc podporujú high level komponenty, ako sú napríklad vytáranie zložitejších tvarov, drag and drop funkcia, používanie paliet a podobne. Oproti nim sa tu nachádzajú aj také knižnice, ktoré podporujú kreslenie low levelových komponentov, ako sú napríklad základné geometrické tvary, čiže kružnice, obdĺžniky, trojuholníky a ostatné. Oba druhy knižníc, či už voľne šíriteľné verejnosti alebo komerčné pre spoločnosti majú svojé využitie. [6]

- **JointJS:** JavaSciptová knižnica pre vytváranie diagramov. Používa sa na vytváranie buď statických diagramov alebo plne interaktívnych nástrojov. Obsahuje kreslenie základných komponentov, ako sú obdĺžniky, kružnice, elipsy, texty, obrázky a cesty. Umožňuje vytváranie vlastných elementov založených na SVG. Obsahuje interaktívne elementy a prepojenia. Výsledný diagram je serializovaný alebo deserializovaný pomocou JSON formátu.

Obsahuje dotykovú podporu. Má vlastnosť priblíženia a oddialenia. Svojimi vlastnosťami je určená na vytváranie elementov známych diagramov pre vizualizáciu vzťahov modelov v ERD, vzťahov zamestnancov spoločnosti v Organization Chart, zobrazenie konečných automatov, UML diagramov, Petriho sietí. [4]

- **Rappid:** Rappid je komerčné rozšírenie predchádzajúcej knižnice JointJS. Ide o skupinu prídavných pluginov a ich komponentov, ktoré rozširujú možnosti modelovania diagramu. Rozširuje úpravu v čase. Obsahuje priamu funkciu kopírovania a vkladania do lokálneho úložiska HTML5. Manipuluje s viacerými elementami súčasne. Má funkciu vrátenia kroku späť a prechodu na nasledujúci krok. Zobrazuje dialógové okná priamo v diagrame. Dokáže vytvoriť automatický tvar orientovaných grafov. Dokáže exportovať diagram vo forme súboru SVG. Je určený predovšetkým pre diagramy UML, ERD, ORG, BPMN. [4]

- **Raphael:** Je to malá knižnica napísaná v JavaScripte, ktorá zjednodušuje prácu s vektorovou grafikou na webe. Dokáže orezávať a otáčať obrázky. Ako základ pre vytváranie grafiky používa SVG W3C odporúčania a vektorový značkovací jazyk (VML). Dokáže zachytávať udalosti alebo ich upravovať. Hlavnou výhodou je poskytnutý adaptér, ktorý umožní, aby bolo vektorové kreslenie kompatibilné s prehliadačom a aby bolo zároveň i jednoduché. [6]

- **D3:** D3 je charakterizované kreslením dokumentu založeného na dátach. Prevádza vizualizáciu dát použitím HTML, SVG a CSS. Dôraz kladie na webové štandardy pre moderné prehliadače. Kombinuje výkonné komponenty vizualizovania a prístup k manipulácii DOM elementov. Poskytuje dynamickú vizualizáciu. Umožňuje nastavenie dynamickej vizualizácie aj pomocou časových razítok, čím sa dynamicky mení prvok samostatne. Umožňuje vytváranie prechodov s animáciami tak, že sa zvolí trvanie animácie, štýl animovania a cieľový bod animácie. Takéto animácie môžu byť zväčšenie, zmenšenie prvku, skosenie, rotovanie a podobne alebo aj zmena parametrov ako sú farba, tvar a iné. Využíva sa na grafy, pretože dokáže vykresľovať štatistiky, povrch, vektory, obrázky.

- **GoJS:** GoJS je JavaScriptová knižnica, ktorá je bohatá na funkcionalitu s kreslením interaktívnych diagramov naprieč modernými prehliadačmi a platformami. Umožňuje vytváranie komplexných uzlov v diagrame, uchovávanie transakčných stavov, kopírovanie a vkladanie, vytváranie paliet, nadhľadov, modelov, diagramov, manipulovanie udalostí, interaktívne ovládanie a podobne. Má prídavné funkcie ako funkciu drag and drop v canvasoch a funkciu krok späť a krok vpred. Je to čisto JavaScriptová knižnica. Kompletne beží v prehliadači a renderuje do HTML5 canvas elementu alebo do SVG objektu bez pomoci zo strany servera. Nie je závislá na ďalších knižniciach alebo frameworkoch. Využíva sa na statické modelovanie, ale aj na interaktívne úpravy. Je určená na komerčné účely za poplatok, ale na skúšanie, nedistribúciu a školské využitie je plne voľná. [9]

Z nášho hľadiska funkcionality jednotlivých knižníc, ich zamerania a licencie je pre nás najlepšou voľbou využitie knižnice GoJS, pretože je primárne určená na vytváranie interaktívnej

tvorby diagramov. Z hľadiska jej funkcií nám poskytuje pevný základ modelovania, ktorý budeme potrebovať. [4]

3.3 Návrh riešenia softvéru

V tejto podkapitole budem vysvetľovať návrh riešenia pre jednotlivé požiadavky. Nájdeme tu návrhy riešenia na vytvorenie aplikácie, návrh rozhrania stránky, návrh databázy a funkcií systému.

3.3.1 Návrh užívateľského rozhrania

Užívateľské rozhranie musí byť intuitívne a mať jednoduché ovládanie, interaktívny prístup a proces modelovania má prebiehať v príjemných farbách. Na nasledujúcom obrázku ilustrujem rozloženie layoutu stránky na modelovanie diagramu aktivít. Správa modelov a diagramov bude prebiehať separátne, avšak na modelovaní sa podieľa diagram aj model súčasne. Hlavné menu bude pozostávať z navigačného panelu na moduly stránky. Bude odkazovať na profil užívateľa, správu modelov pravidiel ontológie a správu diagramov. Jednotlivé moduly budú navzájom poprepájané pomocou hypertextových linkov.

Modelovacie prvky pozostávajú z menu modelovania. Menu modelovania pozostáva z troch možností. Modelovanie podľa pravidiel ontológie tvorí dve metódy - na modelovanie prvkov diagramu alebo na modelovanie závyslosti s komponentami. Tretia možnosť modelovania je na základe vlastných požiadaviek.

Modelovací canvas pozostáva z pozadia, ktoré bude v modernej pastelovej farbe webového dizajnu. Musí byť v prirodzenej farbe a spĺňať všetky potrebné funkcie na modelovanie. V tomto cancase sa budú tvoriť jednotlivé uzly a prepojenia diagramu.

Obrázek 9: Rozloženie layoutu stránky modelovania

Aby modelovanie mohlo prebiehať plynulo, musíme zabezpečiť dostatočne veľký priestor na jeho realizáciu. Pri modelovaní sa zmení rozloženie stránky, zrušia sa odseky zľava aj sprava a roztiahneme kontext stránky a celú zobrazovaciu plochu. Táto plocha sa zobrazí v maximálnych možných rozmeroch, aby užívateľ nemusel skrolovať. Dosiahneme taký efekt, že čím väčší priestor poskytne webový prehliadač, tým väčší bude aj kontext stránky rozdelený na dve časti. Prvú časť tvoria Modelovacie prvky a druhá časť sa skladá z Modelovacieho canvasu. Tieto dve časti sú organizované ako tabuľka, ktorá pozostáva z dvoch stĺpcov. Veľkosť šírky je rozdelená tak, že Modelovacie prvky zaberú 15% poskytnutého priestoru, ale s maximálnou šírkou 300 pixelov. Modelovací canvas si zoberie zvyšných 85%, ale maximálnu hranicu šírky nemá stanovenú. Výška kontextu stránky je taktiež riešená s využitím celej možnej výšky obrazovky, ale s tým rozdielom, že obe časti zaberú 100% výšky. Pod touto tabuľkou sa nachádza prepojenie na index diagramov, teda zoznam všetkých diagramov.

Stránka je rozložená do separátnych modulov, ktoré sa starajú o správu modelov ontológie a diagramov. Využijeme návrhový vzor Model View Controller. Kontroléry majú využitie pre vytváranie, editovanie a mazanie, pretože tu nie je potreba dodávania ďalšej funkcionality. K modelom pridáme validáciu pre jednotlivé elementy modelu.

Pri spravovaní modulu diagramu, jeho vytváraní, úprave a mazaní si vystačíme s klasickým prístupom ako pri modeloch ontológie. Pri editovaní už máme zahrnuté samotné modelovanie na základe zvoleného modelu pravidiel ontológie, ktoré pozostávajú z formalného zápisu. Budeme využívať pravidlá ontológie transformované do formátu JSON.

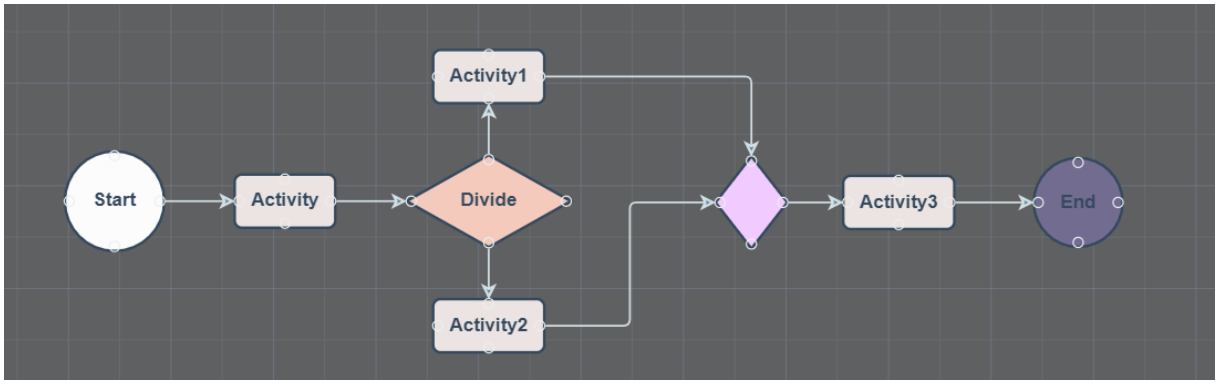
3.3.2 Modelovacie prvky meta-modelu

V tejto práci máme definovaný meta-model, ktorý pozostáva z aktivít, rozhodovacieho uzla, zlučovacieho uzla, počiatočného i koncového elementu diagramu a určenia smeru činností pomocou prepojenia šípok.

Modelovací canvas bude pre lepšiu orientáciu v prostredí pozostávať z mriežky, ktorá bude tvorená štyrmi intervalmi čiar. Dve sa budú nachádzať na osi x a dve na osi y. Najskôr sa nakreslia čiary po intervaloch - každé štyri pixely na osi x aj na osi y, čím sa vytvoria mriežky na pozadí. Následne sa takým istým spôsobom vykreslia čiary po ôsmich pixeloch na oboch osách, ale inej farby, čím dostaneme prehľadnejšie mriežky. Nové mriežky sú väčšie a majú v sebe menšie mriežky. Výsledné pozadie vyzerá modernejšie a pre užívateľa zaujímavejšie. Canvas bude mať tmavšie sivé pozadie, mriežky budú bledších odtieňov sivej a slabomodrej farby, čím sú zvýraznené mriežky. Keďže canvas je tmavšej farby, elementy diagramu budú mať bledšie pozadie, aby boli jasne oddelené a viditeľné vo vzťahu k ostatným prvkom a samotnému pozadiu. Elementy budú mať pre lepšiu organizáciu viditeľnosti zvýraznené obrysy tmavomodrej farby.

Modelovacie prvky sa od seba odlišujú tvarom a identifikátorom, ktorý bude ich menom. Konkrétne ide o opis aktivity, rozhodovacieho uzla a podobne. Aktivity budeme zobrazovať obdĺžnikom béžovej farby so zaoblenými rohmi. Rozhodovacie uzly budú mať slabočervenú farbu, aby charakterizovali rozdielne prevedenie podľa podmienky. Zlúčovací uzol bude charakterizo-

vaný slabofialovou farbou. Začiatočný uzol je zobrazený kruhom, ktorý je vyplnený bielou farbou s textom „Start“. Koncový uzol je tmavomodrej farby s textom „End“. Každý uzol má svoje meno, ktoré je zároveň aj identifikátorom, čím zabránime multiplicite jednotlivých uzlov a dosiahneme jedinečnosť. Prepojenia sú definované pomocou jednostranných šípok sivej farby. Smer šípky charakterizuje tok činností a smer vykonávania procesu. Samotný koniec šípky je zvýraznený vo vnútri bledou farbou. Každý element takto dokážeme odčleniť od ostatných druhov.



Obrázek 10: Grafická reprezentácia meta-modelu aplikácie

Každý uzol má na svojom hornom, spodnom, ľavom a pravom okraji presne v strede guličku vyznačenú bielym obrysom. Tieto guličky sa rozsvietia nasivo, keď sú práve aktívne, teda užívateľ ich má označené alebo je na uzli kurzorom myšky. Jednotlivé uzly sa generujú automaticky, stačí im k tomu iba názov, buď vlastný alebo generovaný podľa pravidiel ontológie. Prvky sa do canvasu pridávajú pomocou udalosti dvojkliku na canvas, konkrétne na súradnice kurzora s preddefinovanú výškou a širkou orientovanej podľa dĺžky textu. V prípade potreby s ním dokážeme manipulovať. Manipulácia prebieha nasledovne. Užívateľ označí konkrétny uzol, klikne naň, ten sa označí obdĺžnikom, ktorý je ohraničený výraznou bledomodrou farbou, čím sa zreteľne odlíši od ostatných elementov. Vyznačený obdĺžnik má svoj obvod kreslený prerušovanou čiarou, po rohoch má vyznačené rohy, ktoré majú funkciu zväčšovania alebo zmenšovania označeného uzlu. Na pravej strane vedľa uzla je bledý kruh, ktorý slúži na rotovanie elementu. Rotovanie prebieha oboma smermi.

Canvas má v sebe vytvorenú funkcionalitu. Pomocou kláves dokážeme ovládať obsah modelovanej plochy.

3.3.3 Správa modelov

Konkrétne ide o správu modelov pravidiel ontológie. Budeme tu riešiť pridávanie nových sád pravidiel, ktoré sú definované v súbore s koncovkou owl. Tieto pravidlá si však musíme najskôr nadefinovať externým, samostatným softvérom s názvom Protége. Protégé je zadarmo, voľne šíriteľný editor ontológie a frameworku pre budovanie inteligentných systémov. Je podporovaný

silnou kominutou akademikov, vládou, korporáciami, biomedikmi, e-comerce systémami a podobne. Má široké využitie.

Program si stiahneme na oficiálnych stránkach, nainštalujeme a môžeme ho už rovno používať. Nie sú potrebné ďalšie registrácie. Pomocou tohto nástroja dokážeme nadefinovať aktivity, roly, postupnosť prevádzania aktivít i zdroje. Výsledný súbor sa uloží vo formáte ontológie. Ide o XML súbor, ktorý má špeciálne značenie, špecifikáciu a väzby.

Následne sa v softvéri prepneme na správu modelov. Základná stránka správy modelov je indexová stránka, na ktorej je zoznam všetkých prístupných modelov užívateľa. Každá položka v zozname má v sebe prepojenie na editáciu, zobrazenie detailov a mazanie. Vyberieme možnosť vytvoriť nový model. Zobrazí sa nám stránka, ktorá ma v sebe povinné parametre, ktoré musíme vyplniť. Vyplníme názov pridávaného modelu, ktorý je jedinečný. Hlavným identifikátorom je však jedinečné identifikačné číslo. Je tu kontrola jedinečnosti názvu, pretože chceme zabrániť multiplicite názvov. Keby sme túto podmienku neochránili, vznikali by tu nekonzistentné názvy, čo by viedlo k zmätku. Následne pridáme súbor ontológie cez dialógové okno prehľadávania súborov.

Tento prehliadač má nastavený filter, aby zobrazoval iba súbory ontológie. Po pridaní sa súbor načíta do obsahu vstupu. Kontext vstupu súboru sa zobrazí do plochy, ktorá reprezentuje HTML textový box. V tomto textboxe je zobrazený celý obsah súboru. Ak sa súbor nenačítal správne, znamená to, že nespĺňa podmienky validácie a do textboxu sa nezobrazí nič. Tento spôsob riešenia je priamym prístupom k dátam hneď po načítaní, pretože všetky moderné prehliadače, ako sú Google Chrome, Opera, Firefox a podobne, z hľadiska bezpečnosti nezobrazujú celú cestu k vybranému súboru cez dialógové okno. Zobrazuje sa iba zdrojové miesto, čiže názov konkrétneho disku. Ďalej pokračuje lomkou a na konci pridá názov súboru s príponou. Medzi súborom a názvom disku je v lomkách zobrazená cesta s menom FAKEPATH. Názorný príklad je "C:\$FAKEPATH\ontology.owl".

Dialógové okno sa vyvolá s HTML elementom s tagom charakterizujúcim vstupné dáta so súborom, konkrétne sa volá INPUT s typom FILE, čo vystihuje súbor. Tento element si dokáže držať a priviazať konkrétny súbor, čiže sa ľahko dostaneme k jeho obsahu. Tento element si dokáže držať a priviazať konkrétny súbor, ľahko sa teda dostaneme k jeho obsahu. Túto vlastnosť som využil a pre lepší prehľad som celý obsah súboru vypísal rovno do textového boxu, s ktorým budem ďalej pracovať a využívať hlavne v modeli a kontroléri.

Po vyplnení povinných parametrov sa klikne na tlačidlo CREATE, ktoré charakterizuje vytvorenie nového modelu. Všetky parametre, ktoré sa vypĺňajú a zobrazujú, sú vo formulári, ktorý je vytvorený pomocou RAZOR syntaxe „@Html.BeginForm", v ktorom sú parametre na volanie kontroleru a špeciálnej akcie v ňom. RAZOR pracuje a komunikuje aj s modelom, čiže dokáže všetky parametre uložiť do modelu View. Tento formulár konkrétne volá na kontroler pre správu modelov ontológie a v ňom na akciu Create. V tejto akcii prebieha ešte validácia modelu. Ak je validácia nesprávna, vráti obsah modelu do stránky pre úpravu a zobrazí chybové hlášky.

V opačnom prípade, ak je model správny, proces pokračuje. Pravidlá ontológie sa nama-

pujú podľa OWL Parseru. Slúži na mapovanie formálnych owl pravidiel. Tento formálny zápis pravidiel sa pretransformuje do skráteného JSON objektu, ktorý obsahuje potrebné dáta na modelovanie a overovanie. Na nasledujúcej ukážke demonštrujem zápis uzlov, objektov, väzieb a závislostí prvkov ontológie vo formáte JSON. V ňom sú základné charakteristiky, ktoré sa využijú pri modelovaní diagramu aktivít.

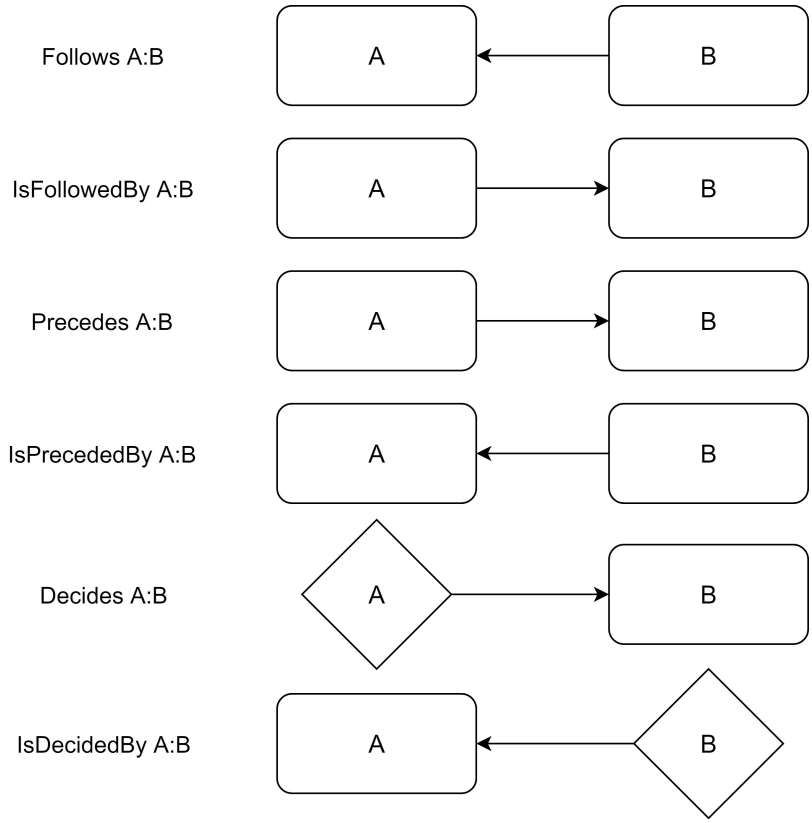
Na základe nášho meta-modelu a obsahu pravidiel ontológie, ktoré sme si vytiahli zo súboru ontológie do nášho JSONu, riešime nasledujúce funkcionality:

- **Uzol Aktivita:** Ide o aktivitu v procese. V ontológií značený pojmom Activity.
- **Rozhodovací uzol:** Ide o rozdelenie možnosti prevádzania procesu. Má pomenovanie Divide.
- **Alternatíny uzol:** Obsahuje možnosti konkrétneho rozhodovacieho uzlu. V modeli ho nájdeme pod pojmom Alternative.
- **Väzba Follows:** Charakterizuje, že v rámci priebehu procesu jeden prvok nasleduje druhý. V ontológií je označená ako Follows. Príkladom je "[#EditDocument]:[#CreateDocument]" a charakterizuje, že aktivita [#EditDocument] v rámci vykonávania procesu prebehne neskôr ako aktivita [#CreateDocument]. Nejde o pevnú väzbu.
- **Väzba IsFollowedBy:** Je to opačný zápis ako pri Follows. Neplatí, že všetky väzby z Follows sa nachádzajú aj vo väzbe IsFollowedBy a opačne, preto sa musia riešiť separátne. Nie je pevnou väzbou.
- **Väzba Precedes:** Charakterizuje ekvivalentnú väzbu ako je IsFollowedBy. Opäť tu platí rovnaké pravidlo, že nie všetky väzby z IsFollowedBy sú aj v Precedes. Daná väzba sa taktiež musí riešiť separátne a nemá pevnú väzbu.
- **Väzba IsPrecededBy:** Opačná väzba ako Precedes, ekvivalentná väzba s Follows. Platí tu stále rovnaké pravidlo, a preto sa taktiež rieši separátne. Nie je to pevná väzba.
- **Väzba Decides:** Táto možnosť charakterizuje stav, že aktivita rieši konkrétny rozhodovací uzol. Je to pevná väzba.
- **Väzba IsDecidedBy:** Ide o opačný zápis ako Decides. Konkrétny uzol je riešený aktivitou. [16]

Pevná väzba znamená, že pri vykonávaní určitej aktivity alebo uzla je pevne stanovené, že uzly nasledujú za sebou presne tak, ako to ukazuje zápis, čiže medzi uzlami nie je žiadny dodatočný uzol. Na druhej strane väzba, ktorá nie je pevná, znamená opak, čiže sa medzi nimi môžu nachádzať ďalšie uzly. Smer vykonávania stále platí. Podľa týchto väzieb sa kontroluje správnosť diagramu. Musí sa skontrolovať každá väzba a všetky pravidlá. Tým dostaneme validáciu podľa

pravidiel ontológie.

Nasledujúci obrázok popisuje grafické znázornenie závyslosti podľa pravidiel ontológie.



Obrázek 11: Grafické znázornenie závyslosti ontológie

V tejto práci vychádzame z toho, že nadefinované pravidlá v súbore ontológie sú namodelované podľa noriem a korektne. Čiže nemôže nastať spätný chod, a teda podľa pravidiel ontológie nemôže byť definovaný stav napríklad: Follows [#CreateDocument]:[#EditDocument], Precedes [#CreateDocument]:[#EditDocument]. Týmto zápisom rozumieme, že aktivita [#CreateDocument] prebehne v rámci vykonávania procesu neskôr ako aktivita [#EditDocument]. Druhý zápis vystihuje opačný stav s rovnakým zápisom, t. j. aktivita [#CreateDocument] sa prevedie skôr ako aktivita [#EditDocument]. Je očividné, že ide o nekorektný zápis pravidiel, tým pádom jedna väzba automaticky vylučuje druhú. Výsledok takejto validácie bude vždy nepravdivý.

Následne ilustrujem JSON pravidiel, vytvorený podľa owl pravidiel.

```
{
"Activities": [ "[#Potvrdit_ulozeni_oceneni]", "[#Zadat_povinne_udaje]", "[#
    Zkontrolovat]", "[#Zobrazit_detail_oceneni]" ],
"Alternatives": [ "[#Stornovat_oceneni]", "[#Ulozit]" ],
```

```
"Decides": [ "[#Stornovat_oceneni]:[#Zkontrolovat]", "[#Ulozit]:[#Zkontrolovat]
    " ],
"Divide": [ "[#Zkontrolovat]" ],
"Follows": [ "[#Zadat_povinne_udaje]:[#Zobrazit_detail_oceneni]", "[#
    Zkontrolovat]:[#Potvrdit_ulozeni_oceneni]" ],
"IsDecidedBy": [ "[#Stornovat_oceneni]:[#Zkontrolovat]", "[#Ulozit]:[#
    Zkontrolovat]" ],
"IsFollowedBy": [ "[#Potvrdit_ulozeni_oceneni]:[#Zkontrolovat]", "[#
    Zobrazit_detail_oceneni]:[#Zadat_povinne_udaje]" ],
"[#Stornovat_oceneni]:[#System]", "[#Ulozit]:[#System]", "[#Zadat_povinne_udaje
    ]:[#Prodavac]", "[#Zkontrolovat]:[#System]", "[#Zobrazit_detail_oceneni]:[#
    System]" ],
"IsPrecededBy": [ "[#Potvrdit_ulozeni_oceneni]:[#Zadat_povinne_udaje]", "[#
    Potvrdit_ulozeni_oceneni]:[#Zobrazit_detail_oceneni]" ],
"Precedes": [ "[#Zadat_povinne_udaje]:[#Potvrdit_ulozeni_oceneni]", "[#
    Zadat_povinne_udaje]:[#Zkontrolovat]", "[#Zobrazit_detail_oceneni]:[#
    Potvrdit_ulozeni_oceneni]", "[#Zobrazit_detail_oceneni]:[#Stornovat_oceneni
    ]", "[#Zobrazit_detail_oceneni]:[#Zkontrolovat]" ],
"Roles": [ "[#Prodavac]", "[#System]" ]
}
```


Výpis 1: JSON objekt definujúci pravidlá ontológie

Modelovací model sa skladá z troch elementov. Prvý z nich je element “class”. Charakterizuje triedu, ktorú tento JSON objekt reprezentuje. Trieda "go.GraphLinksModel"charakterizuje namodelovaný obsah diagramu, teda samotný model. Obsahuje všetky uzly komponenty a väzby diagram.[9]

Nasledujúci element “nodeDataArray” zahŕňa pole všetkých uzlov, z ktorých pozostáva diagram. Každý uzol má svoje špecifické vlastnosti:

- **“key“:** obsahuje kľúč uzla, tento kľuč je jedinečný pre každý uzol
- **“figure“:** zachytáva tvar elementu
- **“id”:** jedinečný identifikátor uzla
- **“fill“:** farba výplne uzla
- **“loc“:** umiestnenie uzla v rámci modelovacieho canvas, pozostáva zo súradníc x a y
- **"width”:** šírka uzla v pixeloch
- **“height“:** výška uzla v pixeloch

- **“angle“:** uhol rotácie uzla v závislosti od jeho stredu
- **“nodeType“:** typ uzla v rámci diagramu aktivít
- **“textNode“:** názov uzla

Posledný element “linkDataArray“ zachytáva všetky väzby a závislosti. Taktiež má svoje špecifické vlastnosti:

- **“from“:** kľúč uzla, od ktorého začína väzba
- **“to“:** kľuč uzla, ku ktorému smeruje väzba
- **“key“:** kľúč väzby
- **“id“:** jedinečný identifikátor väzby
- **“fromId”:** identifikátor uzla od počiatku väzby
- **“toId“:** identifikátor uzla, kde daná väzba končí
- **“textNode“:** názov väzby
- **“points“:** pole bodov, ktoré charakterizujú kontrolné body a zlomy väzby

Pomocou týchto vlastností dokážeme vytvoriť diagram v systéme. Keďže bude obsahovať všetky dané vlastnosti elementov, je potrebné vytvoriť mapovač celého JSON objektu. Uzol “textNode” a “key” obsahujú text uzla s tým rozdielom, že kľúč je identifikátorom, čiže má vždy zadanú hodnotu. Na druhú stranu “textNode” nemusí obsahovať hodnotu, pretože táto hodnota sa vypĺňa do textu uzla. Napríklad pri aktivite má vyplnenú hodnotu a zobrazí text, ale opakom je zlučovací uzol, ktorý nezobrazuje text. Vďaka uzlom diagramu overíme, či namodelovaný diagram obsahuje všetky konkrétne prvky, ktoré sú definované v pravidlách ontológie, teda aktivity, rozhodovacie uzly a podobne. Pri daných uzloch budeme využívať ich kľúč a typ. Kľúč charakterizuje text, ktorý bol doň vložený. Typ charakterizuje typ uzla v aktivitovom diagrame.

Predtým, ako začneme overovať správnosť pravidla ontológie, musime sa presvedčiť, že namodelovaný diagram je namodelovaný správe podľa pravidiel UML. Diagram overuje nasledujúce vlastnosti:

- Existuje počiatočný uzol?
- Existuje koncový uzol?
- Obsahuje štartovací uzol väzbu, ktorá smeruje k nemu?
- Obsahuje koncový uzol väzbu, ktorá začína od neho?
- Obsahuje niektorý uzol, ktorý nie je koncový, menej ako jednu alebo viac ako jednu väzbu, ktorá smeruje od tohto uzla?

- Obsahuje uzol, ktorý nie je spojovacieho typu, viac ako jednu väzbu alebo menej ako jednu väzbu, ktorá smeruje k tomuto uzlu?

Validovanie závislostí sa riadi algoritmom rozkladu diagramu, pretože potrebujeme zaznamenať takú kombináciu ciest, aby sme dokázali zaznamenať všetky uzly. Nie je efektívne riešiť priebeh cyklus, ktorý môže nastať n-krát. Dokážeme detegovať takú kombináciu ciest v diagrame, že pokryjeme celý diagram. Pri každom uzle zaznamenávame, či už bol navštívený, nenavštívený alebo čiastočne navštívený. Navštívený je práve vtedy, ak boli použité všetky možné prechody uzla, čiastočne ak nie sú návštivené všetky prechody uzla, na druhu stranu nenavštívený je vtedy, ak k nemu neprišiel algoritmus. Pri všetkých uzloch je práve jeden takýto prechod okrem zlučovacieho uzla. Tento uzol má spájaciu funkciu väzieb k uzlu, ku ktorému tento zlučovací uzol smeruje. Algoritmus pozostáva z troch vlastností - detekcie hlavnej cesty, spätného prechodu a predného prechodu.

- **Detegovanie hlavnej cesty:** Postupným prechodom od počiatočného uzla, elementov diagramu až ku konečnému uzlu zistíme práve jednu hlavnú cestu. Ak príde algoritmus k navštívenému uzlu, tak sa spätne vráti až k prvku, ktorý je označený ako čiastočne navštívený. Ak v rámci spätného prechodu takýto uzol neexistuje, tak neexistuje ani hlavná cesta v diagrame, a teda nie je možné dostať sa od počiatočného uzla ku konečnému. Algoritmus končí a diagram nie je validný.

- **Back trace (spätný prechod):** Ide o detekciu vedľajšej cesty, ak hlavná cesta už existuje. Spätným chodom sa dostaneme k uzlu, ktorý je čiastočne navštívený, čiže má možnosť prejsť k nenavštívenému uzlu. Algoritmus bude postupne prechádzať nenavštívené uzly až sa dostane k navštívenému uzlu. Cestu od uzla, ktorý bol dekompozíciou označený ako čiastočne navštívený až po prvý uzol, ktorý je označený ako navštívený, si zaznamenáme ako vedľajšiu cestu.

- **Front trace (predný prechod):** Predstavuje detekciu vedľajšej cesty. Nastáva v momente, ak hlavná cesta ešte neexistuje a objaví nenavštívený prvok. Postupným prechodom prvok pokračuje, až sa dostane k uzlu, ktorý je už navštívený. Zaznamená sa cesta od prvého uzla, ktorý bol na začiatku detegovaný ako nenavštívený, až po uzol, ktorý bol detegovaný ako navštívený. Spočiatku prebieha rovnobežne s detekciou hlavnej cesty, ale s tým rozdielom, že hlavná cesta končí až pri koncovom uzle. Front trace končí pri detegovaní navštíveného prvku.

Ak diagram obsahuje hlavnú cestu, prípadne vedľajšie cesty, tak sa overujú pravidlá závyslostí ontológie. Podľa vlastností konkrétnej väzby sa kontroluje indexové poradie uzla v ceste. Napríklad väzba Follow s hodnotou "[Uložiť dokument]:[Vytvoriť dokument]" platí v tom prípade, ak poradový index elementu [Vytvoriť dokument] je menší ako poradový index elementu [Uložiť dokument] v hlavnej ceste. Tieto elementy sa musia nachádzať buď v hlavnej ceste alebo

spoločne na vedľajšej ceste. Ak by sa nenachádzali v spoločnej ceste, tak by to znamenalo, že vďaka jednému z rozdeľovacích uzlov sa nachádzajú na inej možnej ceste a tým pádom nie je možné, aby nastali v rovnakej možnej ceste od počiatku ku koncu.

3.4 Implementácia

3.4.0.1 Použité technologie:

ASP.Net Core 2.0: Voľne šíriteľný webový framework vyvinutý spoločnosťou Microsoft. Je to nadstavba .NET, ktorá je navyše multiplatformová. Modulárny framework, ktorý funguje na oboch .NET Frameworkoch, na Microsofte a na multiplatfrome .NET Core. [15]

Model-View-Controler: MVC je návrhový vzor, ktorý rozdeľuje aplikáciu do 3 skupín komponentov. Sú to komponenty Model, Controller a View. Všetky požiadavky obsluhuje Controller, ktorý pracuje s modelom, nad ktorým prevedie určité úpravy a zobrazí požadované View, kde sú poskytované údaje z modelu. [15]

Razor: Razor je programový model, ktorý je založený na View a prístupu k modelu. Jeho výhodou je jednoduché a efektívne vytváranie webového rozhrania. Razor značenie poskytuje produktívnu syntax pre zobrazenie pohľadov pri MVC webových aplikáciách. [10]

HTML5: Je najnovším vývojovým štandardom pre HTML. Má v sebe nové prvky jazyka, atribúty, rozšírenú sadu technológií. Pomocou tohto štandardu sa budujú nové rozmanitejšej a výkonnejšie webové stránky a aplikácie.

CSS3: Je najnovšia verzia jazyka kaskádových štýlov (CSS), ktorá má rozšírené vlastnosti. Obsahuje zaoblené rohy, nový element video, špecifické prechody a podobne. Taktiež pracuje s animáciami.

JavaScript: JS je interpretovaný programovací jazyk. Využíva sa najmä ako skriptovací jazyk pre webové stránky a aplikácie, kde obsluhuje klientskú stranu. Uplatňuje sa i pri tvorbe nových knižníc. Je to prototypový, multi-paradigmatický, dynamický a objektový jazyk.

Entity Framework Core 2.0 (EF): EF Core je ľahká, rozšírená, multiplatformová verzia Entity Frameworku, ktorá zaobstaráva prístup k dátam. Ide o objektovo-relačnú mapovaciu technológiu, ktorá pracuje s databázovým systémom pomocou objektov .NET. Má v sebe zabudovanú funkciu scaffolding, ktorá slúži na generovanie databázy alebo modelov, kontrolérov a views aplikácie. [10]

GoJS: GoJS je knižnica na vytváranie a kreslenie UML diagramov, ktorú som rozoberal v predchádzajúcej podkapitole. Je založená na HTML, Javascripte a Canvase. [9]

Protégé: Protégé je voľne šíriteľný editor na vytváranie pravidiel ontológie. Je podporovaný silnou komunitou akademikov, vládou, korporáciami, biomedikmi, e-comerce systémami a podobne. Má široké využitie.

Ajax: Ide o JavaScriptový framework, ktorý slúži na asynchornné volanie zo strany klienta na stranu serveru.

Git: Je distribuovaný, voľne šíriteľný verzovací systém, ktorý slúži na spravovanie revízií. Bol vytvorený Linusom Torvaldsom. Využíva sa na malé aj veľké projekty.

Microsoft SQL Server 2014 Service Pack 2: Ide o relačný databázový a analytický systém. Slúži na uchovávanie a poskytovanie dát. Je to databázový systém orientovaný na Microsoft Windows infraštruktúry.

MySql Server: multiplatformný databázový systém od spoločnosti Oracle Corporation. Zameraný hlavne pre Linux, MS Windows operačné systémy. Je to voľne šíriteľný softvér. [11]

Google Cloud cloudový systém od spoločnosti Google. Obsahuje sadu služieb určenú pre cloud computing. Poskytuje služby na ukladanie a analýzu dát.

S pomocou týchto technológií sme schopní vytvoriť aplikáciu na modelovanie a overovanie procesov pomocou UML diagramov, v našom prípade konkrétne diagramu aktivít.

3.4.0.2 Štruktúra programu:

Riešenie projektu je rozdelené do 5 hlavných komponentov, ktoré predstavujú: webovú aplikáciu na vizualizáciu diagramov a dát, knižnicu s obsahom modelov, z ktorých sa skladá diagram, mapovač vlastností modelovaných dát, knižnicu na generovanie diagramu na základne namodelovaného modelu, parser Owl pravidlá Tento parser vyhotovil spolužiak Miroslav Babral.

Komponenty:

- **Webová aplikácia:** Ide o MVC .Net core webovú aplikáciu, ktorá nesie názov UmlModelerWebApp. Je to fundamentálny komponent, v ktorom prebieha hlavná logika programu. Komunikuje s knižnicou, ktorá spravuje modely komponentov diagramu, poskytuje proces modelovania, vytvárania a spracovania modelov, diagramov. V solutione tohto projektu sa nachádza zložka wwwroot, ktorá má v sebe obsiahnuté všetky statické súbory, konkrétne sa tu nachádzajú súbory kaskádových štýlov, JavaScriptové knižnice a ostatné súbory, ktoré sa využívajú na strane klienta. Webová aplikácia pozostáva z MVC architektúry. Logiku zaobstarávajú tri hlavné kontroléry. Všetky komunikujú s Entity Frameworkom Core na komunikáciu s databázovým systémom.

  - **OwlModelController** zaobstaráva spracovanie a realizáciu modelov ontológie, ich ukladanie, vymazanie i editovanie. Konkrétne spolupracuje s asynchrónnym spracovaním ResultAction Index, Details, Create, Edit, Delete, DeleteConfirmed. Pri vymazaní nedovoľuje vymazať model, pokiaľ k nemu existuje aspoň jeden diagram. Najskôr je nutné odstrániť závislé diagramy a až potom samotný model. Pri vytváraní modelov ontológie sa pracuje s obsahom nahravaného súboru, ktorý sa pošle ako parameter v modeli vo formáte UTF-8. Ide o celý obsah XML súboru s pravidlami ontológie. Tieto pravidlá sa namapujú pomocou OWL Parseru. Vyhotovia preddefinované pravidlá na modelovanie procesu, ktorý bude tento model využívať. Ide o spomínaný JSON formát pravidiel, v ktorom sú obsiahnuté všetky uzly a závislosti ontológie.

  - **UmlDiagramsController** spracováva diagramy. Je to hlavný kontorlér, ktorý umožňuje požadované modelovanie. Má obmedzujúce požiadavky pri vytváraní diagramov. Kontroluje, či vytváraný diagram má pridelený model, aby bolo v budúcnosti možné jeho overovanie podľa pravidiel, ktoré sú obsiahnuté vo vybranom modeli. Takisto sú tu asynchrónne ResultAction, ktoré majú funkcie na ukladanie, vymazanie, editovanie. Pri editácií prebieha samotné modelovanie. Pri tomto modelovaní je možné exportovať diagram na dva účely vo forme JSON súboru. Prvým je exportovanie diagramu pre modelovanie. Obsahuje všetky namodelované dáta, závislosti a informácie. Druhým typom exportu je exportovanie diagramu pre účel proces miningu. Tento súbor obsahuje iba základné informácie o uzloch - ako sú kľuč, názov, id, text - a závislostiach, odkiaľ a kam smeruje väzba.
  - **UserAccountController** spracováva užívateľov, ktorí pracujú so systémom, ale pridáva aj nových užívateľov.

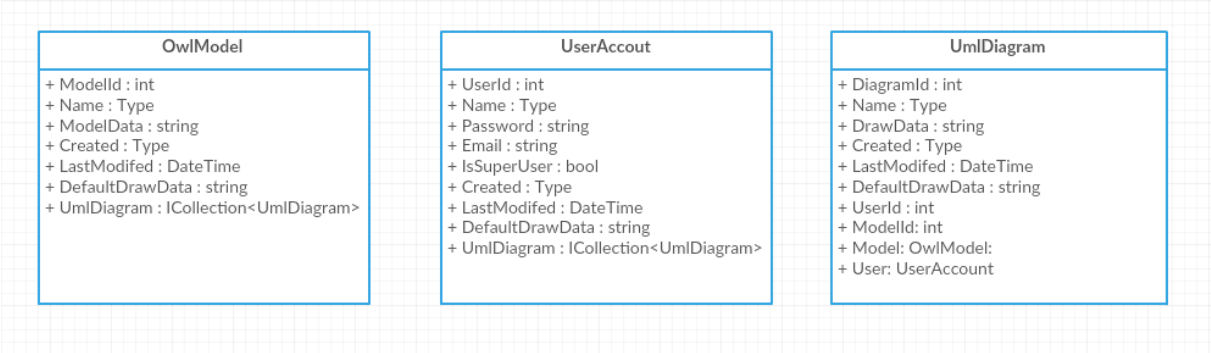
- **UmlDiagramModel:** Predstavuje knižnicu, ktorá zaobstaráva modely diagramu. Model diagramu sa skladá z uzlov a závislostí. Nachádzajú sa tu triedy Diagram, DiagramNode, DiagramPath a podobne.

- **DiagramMapper:** Je to knižnica, ktorá mapuje namodelované dáta z JSON modelu namodelovaného diagramu do štruktúry programu. Podľa exportovaných dát som navrhol štruktúru mapovača tak, aby zachytával všetky atribúty každého elementu v diagrame. Ide o mapovanie JSON stringu do objektu typu Diagram, konkrétne typu DiagramMapper.Diagram. Predstavuje štruktúru namodelovaného diagramu, ktorý sa prevedie do diagramu UmlDiagramModels.Diagram, ktorý sa využíva na riešenie problému validácie.

  Má atribúty, napríklad @class, ktorý podľa definícií frameworku GoJS charakterizuje triedu z diagramu. Ďalej obsahuje List objektov typu NodeDataArray a List objektov LinkDataArray. Trieda NodeDataArray charakterizuje jeden konkrétny uzol v diagrame. Zahŕňa v sebe vlastnosti key, figure, id, fill, loc, width, height, angle, nodeType, textNode. Všetky tieto atribúty už boli popísané. Trieda LinkDataArray reprezentuje konkrétnu väzbu a má v sebe atribúty from, to, key, fromId, toId, points.

- **DiagramGenerator:** Predstavuje knižnicu, ktorá má v sebe implementovanú funkciu GenerateDiagram s parametrom typu string s názvom drawDataModel. Tento parameter nesie v sebe informácie namodelovaného diagramu vo formáte JSON stringu. Tento atribút sa podľa DiagramMapper mapovača deserializuje do diagramu typu DiagramMapper.Diagram. V rámci modelovania sa zvolí možnosť exportu diagramu pre proces mining. Tu sa využije daný diagram, ktorý sa predá tomuto generátoru ako parameter a vyhotoví sa skrip vo forme JSON stringu. Tento výsledok sa exportuje.

- **Model Databázy:** Ako som spomínal, databázový model sa vygeneruje pomocou Entity Framework Core podľa vytvorených modelov v architektúre MVC v solution UmlModelerWebApp. Pre databázový prístup tu máme implementované tri modely.

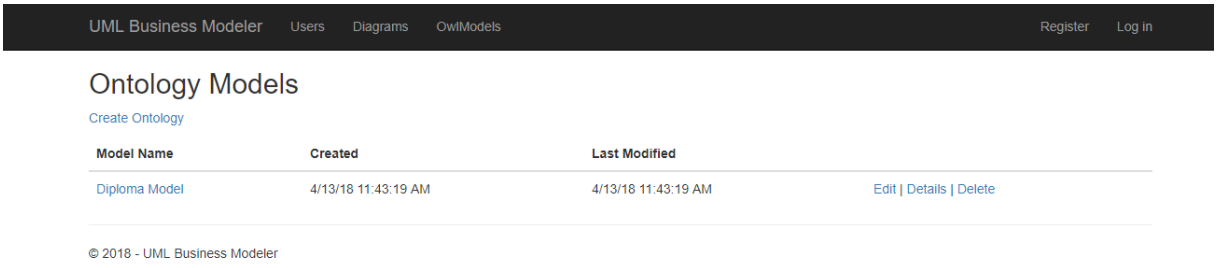


Obrázek 12: Modely MVC architektúry pred generovaním databázy[12]

3.5 Testovanie

Vytvoríme model pravidiel ontológie v nástroji Protégé. Tieto súbory predpripravil spolužiak Miroslav Babral. Využívajú sa pri modelovaní a overení procesu. Práca s Protége je založená na tom, že si ako prvé nadeklarujeme triedy. V našom príade prejdeme do sekcie Entities a položky Classes. Základná trieda je "owl:Thing". Klikneme na túto triedu a stlačíme tlačidlo plus, vyplníme názov triedy "Activity". Potom prejdeme do sekcie Object properties. Je tu základná trieda "owl:topObjectProperty". Klikneme na túto triedu a stlačime tlačidlo plus a pridáme názvy väzieb postupne. Následne sa premiestníme do sekcie Individuals. Stlačime tlačidlo plus a pridáme naše aktivity postupne. Ku každej aktivite možeme po kliknutí na ňu v sekcii Property assertions pridať závyslosť pomocou podsekcie Object property assertions, kde sa vyplní väazba a prvok. Výsledný súbor ukladáme vo formáte OWL/XML Syntax.

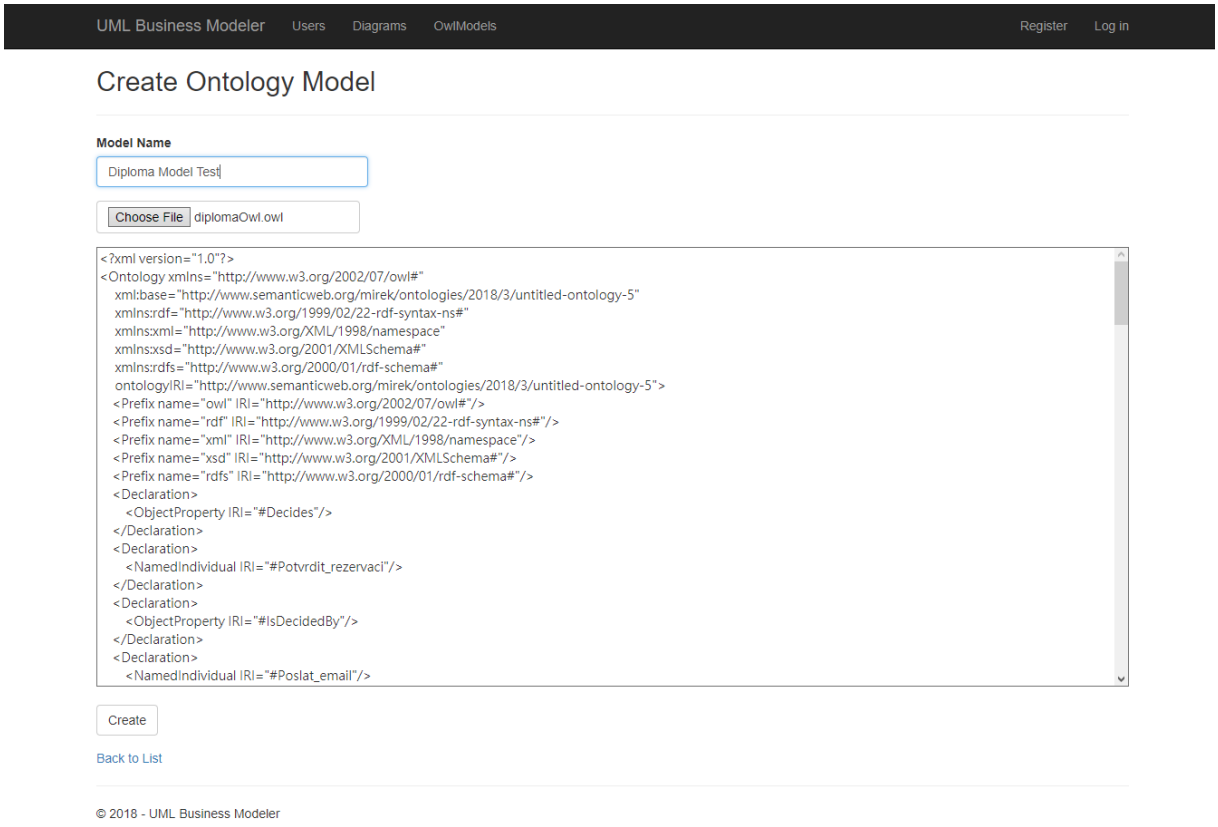
Vo webovom prehliadači zvolíme stránku modelára. Po načítaní programu sa prepneme do sekcie OwlModels. Tu vidíme index, teda zoznam všetkých modelov v systéme.



Obrázek 13: Ukážka modelov pred vytvorením nášho modelu pravidiel

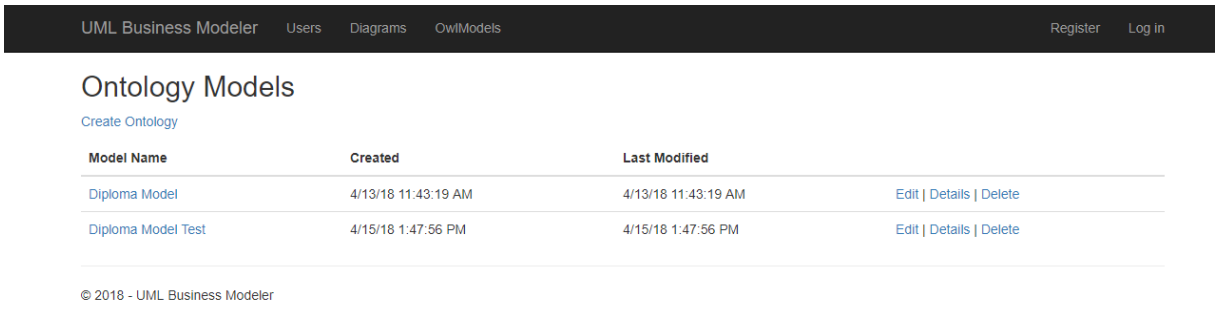
Vyberieme možnosť Create Ontology, prostredníctvom ktorej sa dostaneme k možnosti vytvorenia modelu. V tomto View vyplníme všetky parametre. Vyplníme meno modelu, za pomoci

tlačidla Browse zvolíme súbor typu owl, kde máme vygenerované pravidlá z nástroja Protégé, a stlačíme tlačidlo Create.



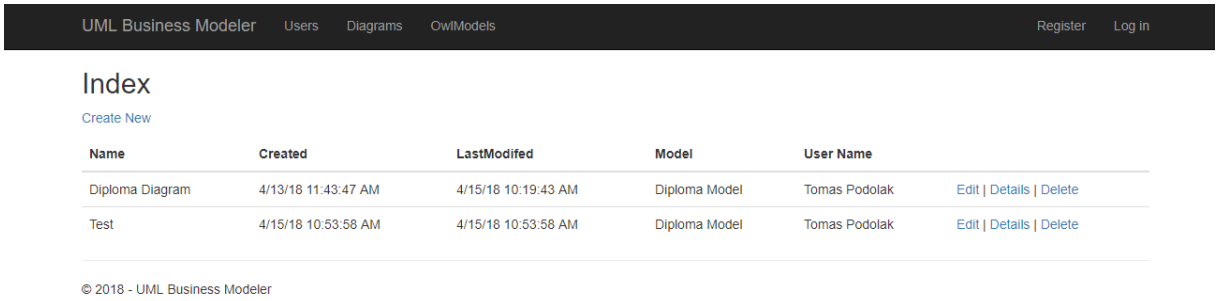
Obrázek 14: Vytváranie pravidiel ontológie

Následne sme presmerovaní do indexu, kde vidíme novopridaný model.



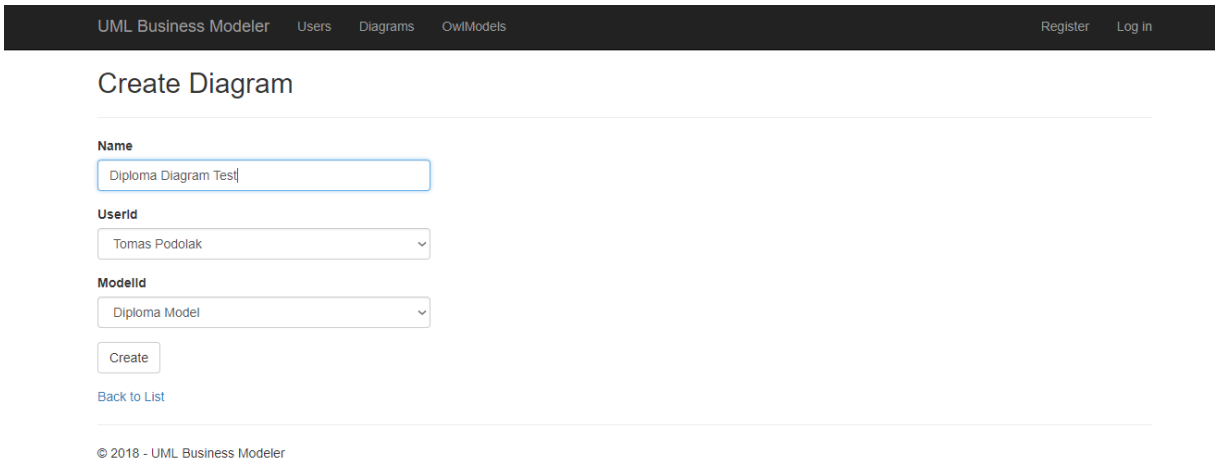
Obrázek 15: Ukážka modelov po vytvorením nášho modelu pravidiel

Princíp je rovnaký, len s inými parametrami. Teraz sa navigujeme do záložky Diagrams, kde vidíme zoznam našich diagramov.



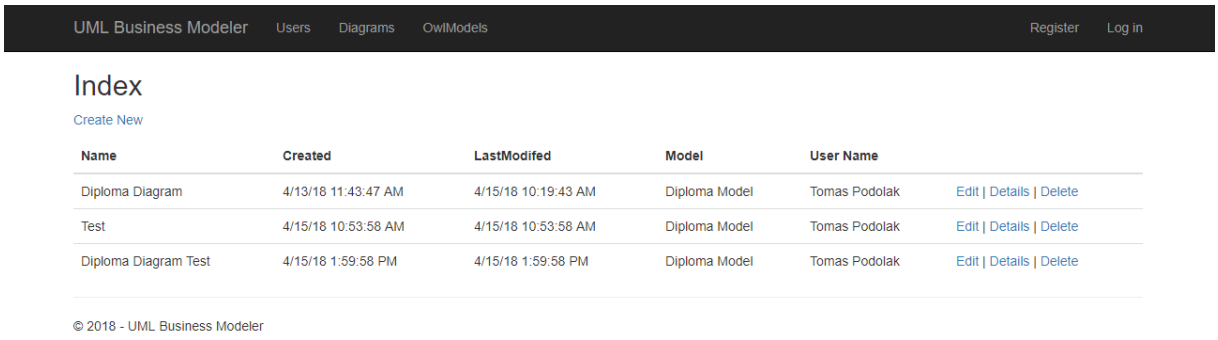
Obrázek 16: Ukážka diagramov pred vytvorením nášho diagramu

Klikneme na možnosť Create New. Zobrazí sa View, ktoré poskytuje rozhranie pre vytváranie diagramov. Vyplníne povinné údaje ako sú názov diagramu, taktiež je potreba priradít užívateľa a model k danému diagramu. Následne klikneme na možnosť Create.



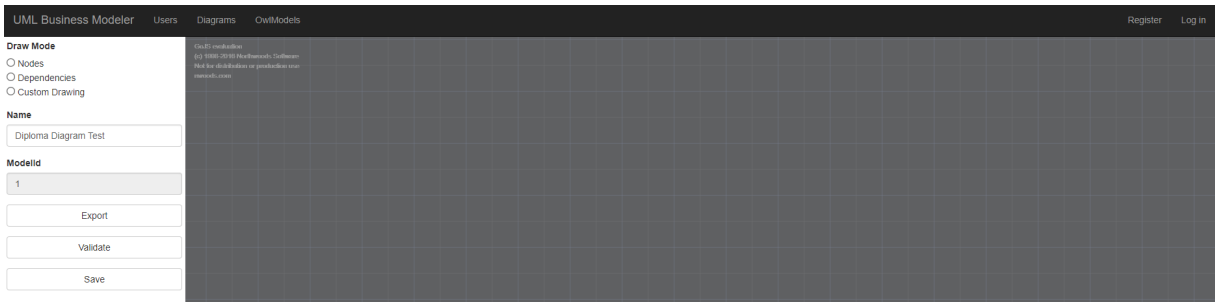
Obrázek 17: Vytváranie diagramov

Teraz sme opäť presmerovaní do indexu diagramov, kde vidíme náš novovytvorený diagram.



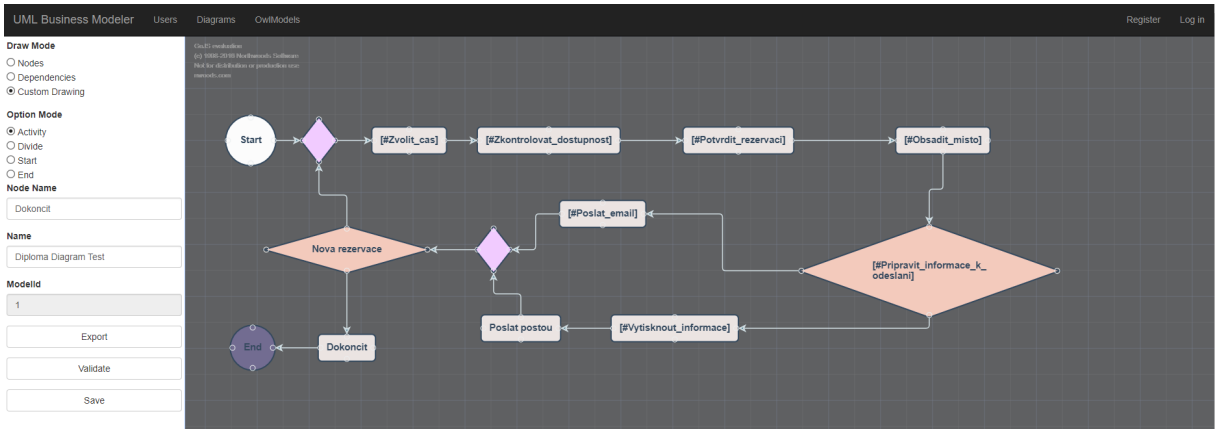
Obrázek 18: Ukážka diagramov po vytvorením nášho diagramu

Teraz už môžeme začať modelovať. Vedľa nášho diagramu klikneme na možnosť editácie, pomocou ktorej sa dostaneme do modelovacieho View. Na začiatku modelovania je náš diagram prázdny. Riadíme sa princípami modelovania pomocou prepínania módov na modelovanie.



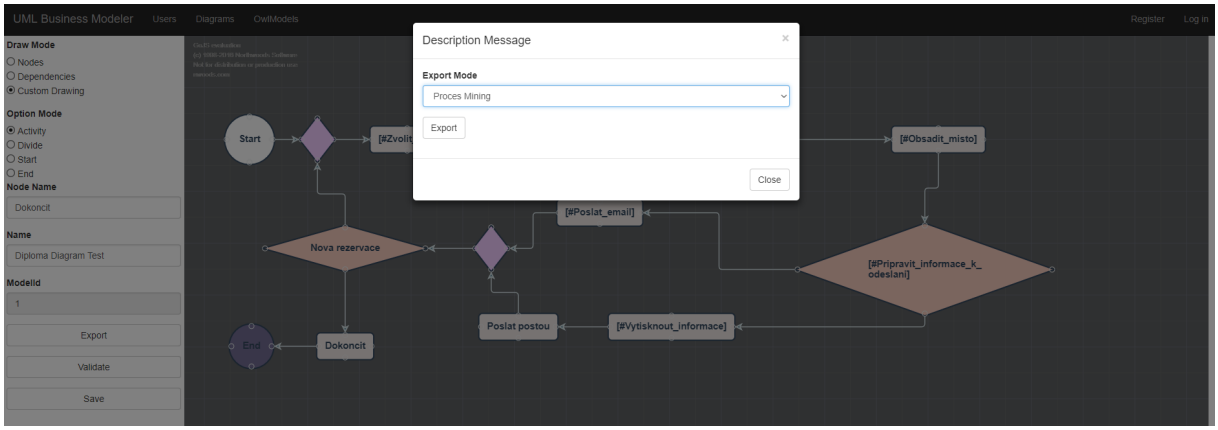
Obrázek 19: Ukážka diagramu na začiatku modelovania

Náš testovací diagram vyzerá podľa pravidiel ontológie vybraného modelu nasledovne.



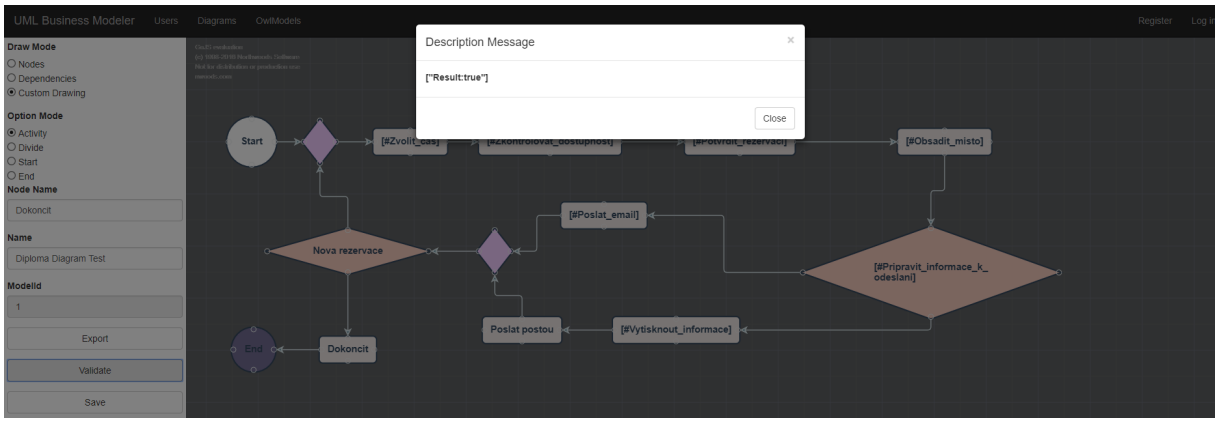
Obrázek 20: Ukážka diagramov po modelovaní podľa pravidiel ontológie

Vľavo máme na výber tri možnosti.Diagramy môžeme ukladať, exportovať alebo validovať. Vyberieme možnosť exportu diagramu a zobrazí sa nám menu, kde zvolíme typ exportu diagramu a vyberieme možnosť Export.

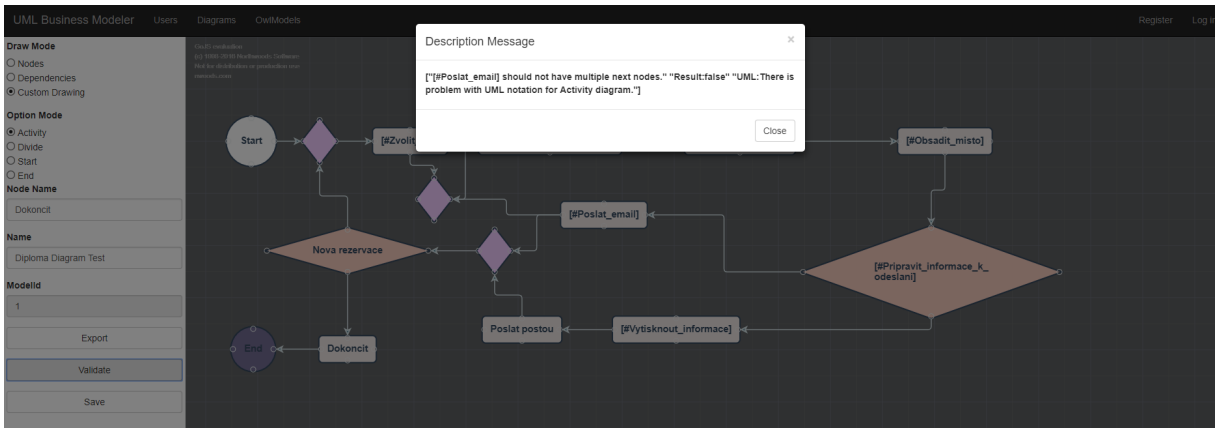


Obrázek 21: Ukážka možnosti exportu diagramu

Teraz vyberieme možnosť validovať a zobrazí sa výsledok hlášok, ktoré vyhodnotil algoritmus. V prípade úspechu napíše "Result:True". V prípade neúspechu zobrazí chybové hlášky.



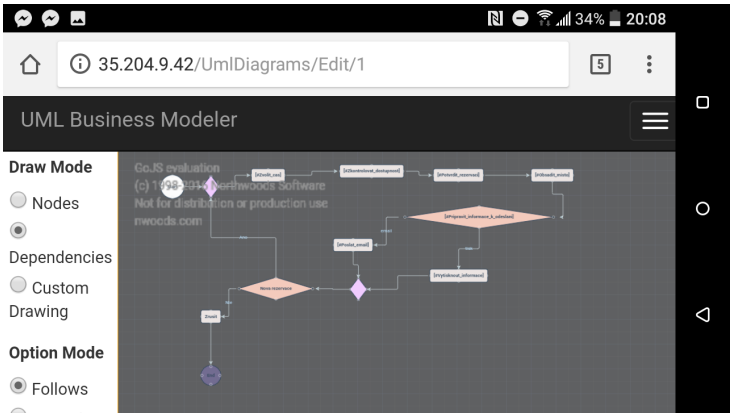
Obrázek 22: Ukážka úspešného modelovania a validovania diagramu



Obrázek 23: Ukážka neúspešného modelovania a validovania diagramu

Všetky zmeny v diagrame sa po vyvolaní možnosti Save uložia a pri ďalšom načítaní sa zobrazí namodelovaní proces.

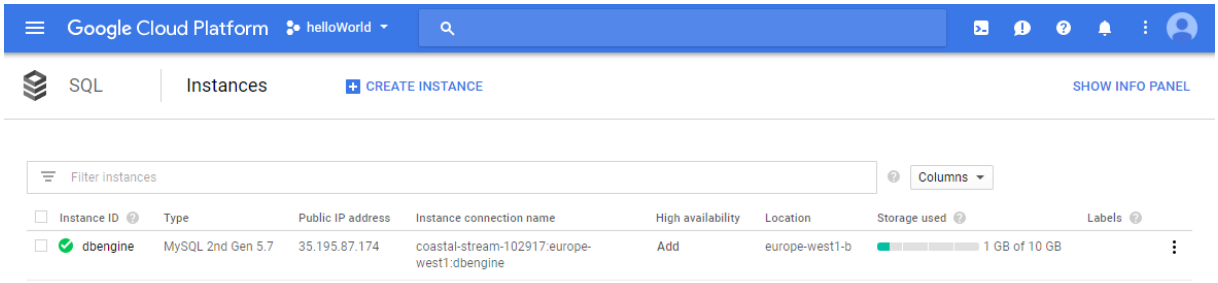
Aplikácia je vďaka navrhnutému dizajnu stránky plne responzívna a rovnaký spôsob modelovania prebieha aj v rozdielnych zariadeniach. Nasledujúci obrázok zobrazuje proces modelovania rovnakého diagramu pomocou smartfónu HTC Desire 650.



Obrázek 24: Responzívne modelovanie na smartfóne

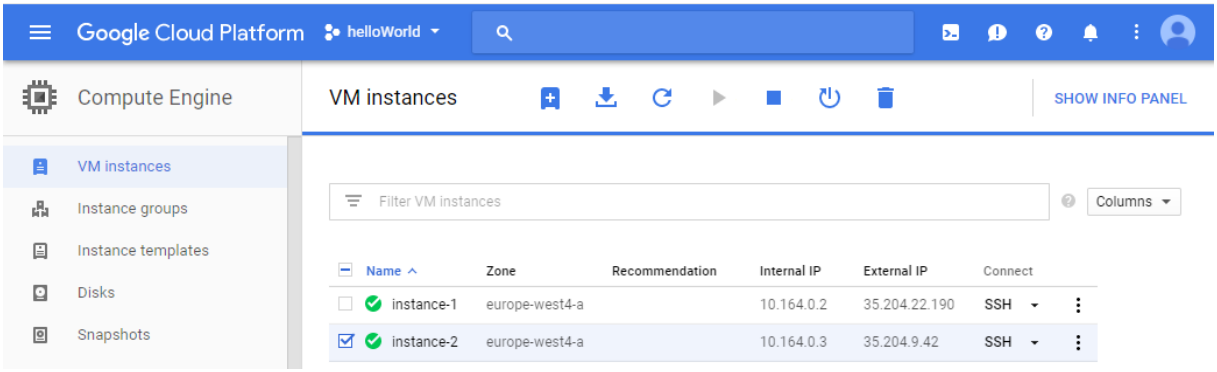
3.6 Nasadenie

Aplikácia je na určitý čas nasadená na serveri Google Cloud. Z tejto technológie sme vytvorili voľný účet na testovanie a vyhodnocovanie nášho nástroja. Urobili sme tu projekt, v ktorom sme vytvorili databázový systém, ktorý je MySql 2nd Gen 5.7. Základné úložisko má 1 GB, no v prípade potreby je možné navýšenie. Pomocou nastavenia firewallu sme korigovali priepustnosť pre všetkých užívateľov. Nastavil sme i databázu ModelerDB a hlavného užívateľa s menom root. Základné špecifikácie zobrazuje nasledujúci obrázok. [11]



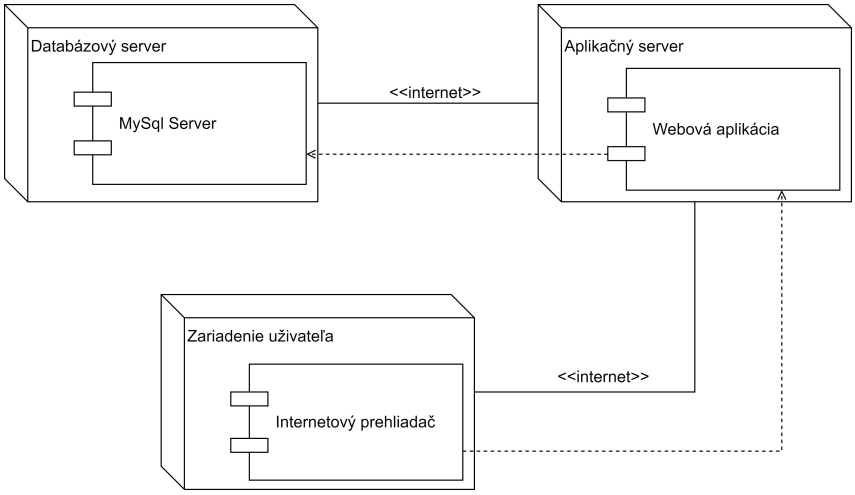
Obrázek 25: Ukážka našej databázy na Google Cloud

Samotnú aplikáciu sme nasadali tiež pomocou technológie Google Cloud. Do rovnakého projektu s databázou sme špecifikovali novú Compute Engine a vytvorili novú VM Machine. Aplikácia bola prevádzaná na systéme Debian 8 i na systéme Debian 9, no pre lepšie výsledky sme ponechali iba verziu prevádzanú na systéme Debian 9. Na daný server sme publikovali aplikáciu, ktorá je dostupná na 35.204.9.42. Nasledujúci obrázok popisuje základné vlastnosti. [11]



Obrázek 26: Ukážka hostingu našej aplikácie na Google Cloud

Diagram komponentov popisuje riešenie aplikácie. Táto aplikácia komunikuje cez webový prehliadač s aplikačným serverom na Google Cloud, ktorý komunikuje s MySql databázou na Google Cloud.



Obrázek 27: Diagram komponentov aplikácie

4 Popis modelovacej časti

4.1 Modelovacia knižnica

Najdôležitejšou časťou systému je modelovanie biznisových procesov. Vytvoril som JavaScriptovú knižnicu UmlDiagramModeler.js. Je založená na knižnici GoJS. Týmto krokom sme dodali nášmu nástroju modelovaciu funkcionalitu. Daný súbor má veľkosť 22KB a obsahuje celú logiku modelovania. Celá funkcionalita je zapuzdrená v objekte, ktorý obsahuje v sebe funkciu, ktorá zavolaním vytvoreného skriptu vytvorí modelovací canvas so všetkými funkcionalitami, ktoré rozoberiem v nasledujúcich riadkoch.

Knižnica po zavolaní ako prvé vytvorí modelovací priestor, teda v html stránke pridá do elementu modelovací canvas, kde prebiehajú všetky modelovacie úpravy. Tento canvas pozostáva z mriežok, ktoré slúžia na lepšiu orientáciu v priestore. Pridal som základnú funkcionalitu, ktorá definuje, ako sa má canvas v určitých prípadoch správať. Udalosti myšky som rozdelil nasledovne. Dvojklikom ľavého tlačidla myšky sa vytvárajú a následne vykresľujú modelovacie prvky na canvas. Kliknutím pravého tlačidla myšky sa vytvorí z modelovacieho diagramu SVG obrázok, ktorý sa posiela na tlačenie dokumentu. Definoval som správanie canvasu tak, že v prípade vytvárania väzieb sa nedá nakresliť väzba, ktorá nemá počiatočný alebo koncový bod. Rovnakú funkcionalitu som definoval aj pre úpravy už existujúcej väzby. Využil som funkcionalitu UndoManagera, ktorá klávesovými skratkami dokáže vrátiť alebo pridať posledné úpravy. Umožnil som animovanie diagramu, kde som nastavil parameter trvania animácie na 3000 milisekúnd. Animácia sa využíva pri modelovaní alebo načítaní diagramu. Diagram dokáže namodelované dáta uložiť ako JSON objekt a načítať diagram z JSON objektu.

Vytvoril som template - vzor pre uzly diagramu. Tento template pozostáva z toho, že celý uzol je zaobalený elementom, ktorý sa nazýva "go.Node". V tomto elemente sú zachytené všetky vlastnosti. Obsahuje funkcionalitu, že daný uzol sa dá dynamicky zväčšovať , rotovať a presúvať. Obsahuje tvar elementu, ktorý sa pre jednotlivé typy uzlov mení. Po stranách má vytvorené porty, ktoré sa pri prechode myškou zvýrazňujú. Porty slúžia na prepojenie uzlov. Template má v sebe zapuzdrené vlastnosti, ktoré sa bindujú pri vkladaní a dajú sa meniť podľa potreby. Tento objekt sa vytvára ako prvok z listu uzlov, teda konkrétne NodeDataArray. V predchádzajúcich kapitolách sme si vysvetlili, ako sa prvky vkladajú pomocou JSON objektu. Hodnoty z tohto JSONu sa bindujú do tohto prvku.

Vytvoril som template aj pre väzby, kde som definoval, ktorá strana väzby je počiatočná a ktorá je konečná. Tieto väzby sa pripájajú na porty uzlov. Linky sa lámu na miestach, kde zmenia svoj smer, väzby sa dajú upravovať. Na konci väzby je šípka, ktorá udáva smer vykonávania procesu. Väzby sú zapuzdrené v objekte typu "go.Link". Väzby, ktoré sa medzi sebou krížia, sa upravia tak, že jedna z väzieb vizuálne preskakuje tú druhú a pri miestne stretu väzieb si vytvorí kopček.

Neimplementoval som funkciu, ktorá dokáže vykresľovať buď samostatné uzly alebo aj celé

väzby. V kóde knižnice sa uchovávajú dva parametre, ktoré v sebe zachytávajú typ kreslenia a kľúč kreslenia. Typom kreslenia rozumieme stav, že v momentálnej situácii počas modelovania sa rozlišuje, či sa kreslí aktivita, rozhodovací uzol alebo celá väzba follow, precedes a podobne. Kľúč obsahuje text modelovacieho prvku. Ak sa vykresľuje uzol, tak tento text pozostáva iba z názvu uzla. Ak sa vykresľuje väzba, tak má v sebe uzly oddelené dvojbodkou. Vo vykresľovanej funkcii sa zistí aktuálny mód a kľúč vykresľovania. Ak ide o väzby, daný kľuč sa rozdelí na dva prvky, ktoré sú uložené v poli. Každému vytvorí špecifický JSON objekt na vkladanie uzlu a skontroluje, či sa daný uzol už v diagrame nenachádza. Ak nie je prítomný, tak ho pridá, a ak áno, tak pokračuje vo vykonávaní funkcie a tento uzol nepridá. Podľa typu väzby sa určí poradie uzlov vo väzbe, to znamená, že sa určí, ktorý uzol je počiatočný a ktorý konečný. Taktiež tu prebieha aj kontrola existencie väzby. Multiplicita nie je povolená.

Diagram zachytáva udalosti, ktoré prebiehajú v diagrame. Pri úprave a vkladaní uzlov sa kontroluje, či niektorý uzol nezmenil svoje meno. Ak prišlo k zmene jeho mena, je potrebné zmeniť jeho kľúč a všetky potrebné atribúty. V tomto prípade ide už o úplne nový uzol, pretože sa zmenia aj väzby, ktoré ho charakterizujú.

Ak diagram detekoval, že sa vytvorila väzba, musí k danej väzbe pridať potrebné parametre, ako sú identifikačné čísla uzlov a podobne. Pri zmene väzby upravuje tieto vlastnosti. Ak došlo k spomínaným udalostiam alebo k úprave väzieb, tak sa prevedie algoritmus, ktorý detekuje, či niektorý z uzlov nemá viac ako jednu väzbu, v ktorej je reprezentovaný ako koncový uzol. Ak áno, tak vytvorí k danému uzlu pripájací uzol, ktorý zlúči všetky jeho pripájacie väzby. Tieto väzby budú smerovať k tomuto podpornému uzlu a samotný podporný uzol bude smerovať k pôvodnému uzlu.

Je nutné podotknúť, že všetky úpravy musia prebiehať v rámci transakcií, ktoré prevádza knižnica GoJS. Každá transakcia sa začína príkazom, ktorý sa volá z diagramu a metódy "StartTransaction", po úspešnom prevedení transakcií diagramu sa volá podobným prístupom aj metóda "CommitTransaction"a v prípade neúspešného prevedenia transakcie sa volá funkcia "RollbackTransaction". Je to z bezpečnostného dôvodu modelovania. V prípade, ak nastane chyba v rámci transakcie, diagram sa vráti do funkčného stavu pred začatím vykonávania tejto transakcie.

Nasledujúca tabuľka demonštruje niektoré vybrané funkncie modelovacieho canvasu.[9]

Príkaz	Funkcia	Popis
Delete, Backspace	Vymazuje pri označovaní prvky	Označenie prebieha udalosťou vybratia uzlu myškou, pomocou ľavého tlačidla
Ctrl-C, Ctrl-Insert	Kopíruje označený uzol	Ozol sa označuje ľavým tlačidlom myšky
Ctrl-V, Shift-Insert	Vkladá skopírované uzly	Musí byť canvas aktívny
Ctrl-A	Označenie všetkých uzlov v diagrame	Modelovanie musí byť aktívne
Ctrl-Z, Alt-Backspace	Vráti poslednú úpravu v diagrame späť	Funkcia Undo.
Ctrl-Y, Alt-Shift-Backspace	Vráti sa na nasledujúcu úpravu	Dá sa volať, ak sme vyvolali akciu upravovania posledných zmien. Funckia Redo.
Up	Posun v canvase nahor	šípka na klávese smerom nahor
Down	Posun v canvase nadol	šípka na klávese smerom nadol
Left	Posun v canvase vľavo	šípka na klávese smerom vľavo
Right	Posun v canvase vpravo	šípka na klávese smerom vpravo
-	Zmenší zoom level	Zoom Out (oddialenie)
+	Zväčší zoom level	Zoom In (priblíženie)
Esc	Zruší aktuálnu vykonávaciu akciu	V prípade, ak sa nevykonáva žiadna akcia, ostáva sa v aktálnom stavo.
Shift-Z	Fit Zook	Vypočíta a nastaví zoom level tak, aby bolo vidno celý diagram
PageUp	Scrolovanie v diagrame smerom nahor	Posun v diagrame je podľa konštanty
PageDown	Scrolovanie v diagrame smerom nadol	Má konštantný posun

Tabulka 1: Zoznam vybraných vlastností canvasu

4.2 Proces Modelovania

Modelovanie prebieha v troch módoch. Každý mód má svoje nastavenie, ktoré si vytvorí možnosť, podľa ktorej sa bude vykresľovať. Dajú sa kedykoľvek kombinovať, čiže nie sme závislí od jedného druhu. Módy majú rozdielnu funkcionalitu a zameranie. Vykreslenie prebieha pomocou vyvolania udalosti dvojkliku v modelovacom canvase. Táto funkcia nám zachytí súradnice kurzoru v priestore. Na týchto súradniciach sa vykreslí konkrétny uzol. Kreslenie prebieha buď manuálne alebo automaticky. Manuálne myslíme, že vyplníme atribúty a vykreslíme dvojklikom na editačnú plochu konkrétny uzol. Automatickým kreslením rozumieme výber jednej z možností, ktorá sa zobrazí podľa módu a pomocou dvojkliku sa nám vykreslí väzba podľa pravidiel ontológie a nie jeden prvok. Pri pridávaní uzlu do plochy sa vždy skontroluje, či daný uzol už existuje. Kontrola prebieha podľa názvu, ktorý je zároveň aj jedinečným identifikátorom. Zaistíme, že názov a zároveň aj identifikátor sú jedinečné pre každý element diagramu.

Spájanie uzlov je aktívne pri aktivite kurzora na uzle. Rozsvietia sa nám okraje uzla a zvýrazní sa guličkа na každej strane v strede. Tieto guličky charakterizujú začiatok alebo koniec väzby. Pomocou myšky sa cez tieto body začína kreslenie väzieb. Šípka má spočiatku bledomodrú farbu. Na začiatku a na konci šípky je šípka označená ružovou farbou. Takto značíme začiatok a koniec väzby. Šípka mení veľkosť a smer podľa kurzora. Môže smerovať na konkrétny uzol alebo na jeho konkrétny vstupný bod. Prístupné vstupné body sú zvýraznené počas modelovania ružovou farbou, čím zvýrazníme možný vstup. Ak smerujeme rovno na uzol a nie na jeho vstupné body, vypočíta sa najkratšia cesta medzi uzlami a spoja sa v najbližších vstupných bodoch. Každý uzol ma tri vstupné body a jeden výstupný. Do výstupného bodu sa nedá pripojiť. Pri vytváraní väzby od výstupného bodu sa smer šípky otočí a začína kreslenie od výstupného bodu k vstupnému. Namodelovanú šípku môžeme zmeniť tak, že na ňu klikneme, na konci a začiatku sa nám rozsvietia guličky, ktoré charakterizujú pripojenia. Stlačíme jeden z týchto bodov a zmeníme väzbu.

Zápis väzby je charakterizovaný objektom, ktorý obsahuje parameter FROM a TO. Ide o počiatočný a koncový uzol väzby, konkrétne o ich identifikátory. Pri pridávaní sa kontroluje, či už existuje väzba, teda či existuje prvok medzi väzbami, ktorý má rovnaký spájací a koncový uzol.

Modelovacie módy:

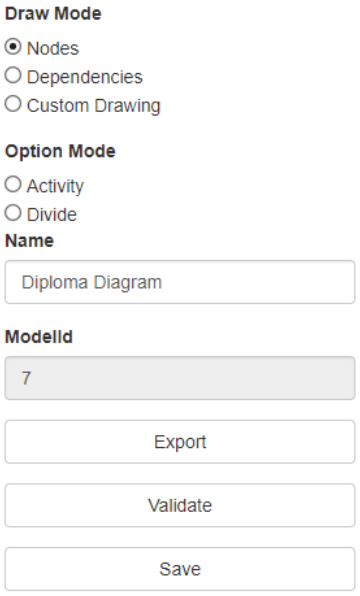
1. Nodes
2. Dependencies
3. Custom Drawing

Nodes mód má za úlohu kresliť iba uzly, ktoré sa nachádzajú v pravidlách ontológie. Vykreslí konkrétny uzol. Neobsahuje možnosť kreslenia počiatočného a koncového uzlu.

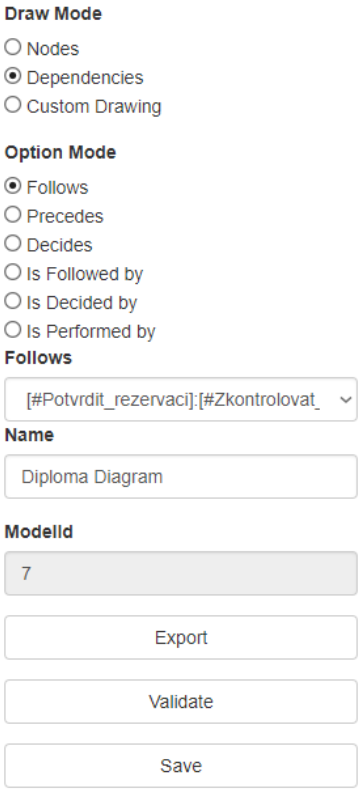
Dependencies mód zahŕňa vykresľovanie konkrétnej väzby, ktorá je definovaná v pravidlách. Keďže zápis pravidiel sme si prispôsobili a vieme, že uzly ontológie sú v hranatých zátvorkách, ich meno začína mriežkou a uzly sú v zápise závislosti oddelené dvojbodkou. Toto dokážeme využiť ako identifikátor rozdeľovania väzby. Zápis väzby rozdelíme podľa charakteru dvojbodky a máme dva názvy prvkov. Vyvoláme vykreslenie, program rozdelí väzbu a najskôr pridá prvý prvok a potom druhý. Podľa závislosti pridá aj väzbu. Prvky sa vykreslia pod sebou a sú pripravené, aby si ich užívateľ premiestnil podľa potreby.

Custom Drawing mód, v preklade vlastné kreslenie, slúži na vytváranie a kreslenie uzlov diagramu aktivít, ktoré nie sú obsiahnuté v pravidlách ontológie. Je primárne určený na tvorbu aktivít a rozhodovacieho uzlu. Jeden zo zmienených uzlov sa vyberie ako možnosť a zobrazí sa pole, do ktorého sa zadá názov prvku. Následne sa podľa kurzora vykreslí bod na plochu. Kreslí sa vždy iba jeden uzol. Pomocou tejto voľby vykresľujeme uzly, ktoré nie sú zahrnuté v pravidlách. Pridávame vlastné uzly a väzby. Iba v tomto móde je možné vykresliť počiatočný a koncový uzol diagramu, preto sa bez neho nezaobídeme.

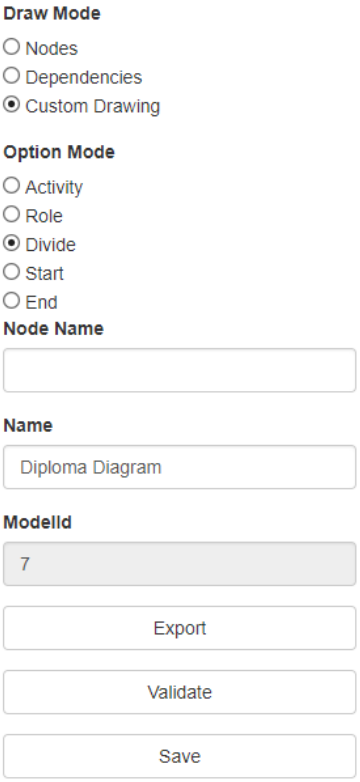
Modelovanie je hlavnou prioritou tejto diplomovej práce, preto musíme dokázať zaobstarať takú funkcionalitu modelovania, aby prebiehala intuitívne, jednoducho, čo možno najviac automaticky, predpovedala závislosti, poskytovala príjemné prostredie a korektnú logiku. Kombináciou týchto módov dokážeme zaobstarať tieto požiadavky. Diagram má slúžiť na ďalšiu analýzu alebo na návrh.



Obrázek 28: Vykresľovací mód Nodes



Obrázek 29: Vykresľovací mód Dependencies



Obrázek 30: Vykresľovací mód Custom Drawing

Pod modelovacím nastavením sa nachádza názov diagramu a modelu, podľa ktorého prebieha modelovanie. Následne sa tu nachádza tlačidlo na uloženie súčasného stavu diagramu a jeho zvalidovanie.

Základné zásady modelovania:

- pre začatie modelovania musím mať vytvorený diagram pozostávajúci z modelu a užívateľa
- vykresľujem počiatočný a koncový bod v móde Custom Drawing
- modelovanie pravidiel ontológie prebieha v módoch Nodes a Dependencies
- aby bol diagram validný, musí obsahovať všetky pravidlá ontológie
- maximálna dĺžka mena uzlu je 20 znakov a minimálna 2 znaky
- modelovanie vlastných uzlov prebieha v móde Custom Drawing
- manuálne vykreslenie väzieb kreslím kliknutím od uzla k uzlu a nie do miesta, kde nie je žiadny uzol

- validovať diagram môžem počas modelovania hockedy a niekoľkokrát
- nekreslím väzby, ktoré vedú k tomu istému uzlu, od ktorého väzba začína
- využívam čím možno najviac plochy k modelovaniu
- nevyužívam názvy uzlov ako “Start” a “End”, tieto označenia slúžia na pomenovanie počiatočného a koncového uzlu

Výsledný modelovací model má nasledujúci formát:

```
{
  "class": "go.GraphLinksModel",
  "nodeDataArray": [ { "key": "Start", "figure": "Circle", "id": 2, "fill": "#
     fcfcfc", "loc": { "class": "go.Point", "x": -258.6365, "y": -314.0152 },
     "width": 50, "height": 30, "angle": 360, "nodeType": "Start", "textNode":
      "Start"} ],
  "linkDataArray": [ { "from": "[#Zadat_povinne_udaje]", "to": "[#
     Zobrazit_detail_oceneni]", "key": "", "id": 7, "fromId": 4, "toId": 5, "
     textNode": "", "points": [ 166.0252, -314.0252, 176.0272, -313.0222 ] } ]
}
```

Výpis 2: JSON zachytávajúci modelovací model

5 Vyhodnotenie

Podrobné štúdium sme zamerali na problematikou biznis procesov, čo je ich účelom a k čomu slúžia. Zistili sme, že každý podnik potrebuje pre lepší prehľad a orientáciu vizualizovať svoje procesy. Zobrazením procesov umožníme podniku vykonávať svoju prácu efektívnejšie a zároveň aj zvýšiť progres. Aby sa proces mohol meniť a prípadne zlepšovať svoje prevedenie, musí byť korektne namodelovaný. Nástroje, ktoré sa zaoberajú touto tematikou, majú rozličné prístupy k riešeniu problémov, ich primárny cieľ je spoločný. Týmto fundamentálnym cieľom je poskytnúť, čo možno najlepšie podmienky pre samotné modelovanie tak, aby bol užívateľ sám schopný modelovať diagramy. Zámerom je uľahčiť chápanie podnikového procesu a poskytnúť obraz ako podklad pre analýzu procesu.

Taktiež sme analyzovali aj knižnice, pomocou ktorých sa dané nástroje tvoria. V tejto práci sme sa vydali cestou modelovania procesov pomocou modelovacieho jazyka UML. Spravili sme prehľad dostupných knižníc, ktoré sme mohli použiť a vybrali sme si tú, ktorá našim požiadavkám vyhovovala najviac. Aby sme zachytili proces, orientovali sme sa na dokumentáciu modelovacej knižnice. Dokázali sme vytvoriť požadovaný nástroj, ktorý umožňuje modelovanie procesov a ich spätné overovanie. Zabezpečili sme vytváranie elementov diagramov, konkrétne diagram aktivít. Predtým, ako sme začali modelovať, potrebovali sme spracovať modely pravidiel. Ontológia má rozsiahle parametre. Vďaka nástroju Protégé dokážeme vytvárať súbory ontológie. Práve tieto súbory spracovávame v systéme. Sme schopní udržať ich na serveri v celej podobe. Pre modelovanie a overovanie sa tieto pravidlá upravujú tak, aby sa zachytili hlavné väzby, ktoré sa využívajú v rámci nášho zvoleného konkrétneho diagramu. Validácia a modelovanie prebieha na jednom rozhraní. Po vytvorení modelu ontológie sa vytvoria pravidlá uzlov a väzieb. Diagram sa postupne modeluje a pri overovacom procese sa opäť riadi podľa vytvorených pravidiel uzlov a väzieb, čím sme zabezpečili spoločné pravidlá modelovania a overovania. Pri modelovaní je nevyhnutné riadiť sa zásadami, ktoré som v tejto práci opísal.

Vytvorili sme nástroj, pomocou ktorého dokážeme modelovať dané podnikové procesy a následne ich overovať podľa preddefinovaných pravidiel ontológie, ktoré si užívateľ vytvorí samostatne. Tieto pravidlá sú nám poskytnuté vďaka programu Protégé. Naše riešenie pozostáva z týchto krokov – vytvorenie modelu ontológie, uloženie modelu do systému, vytvorenie diagramu a priradenie modelu, editovanie a modelovanie diagramu. Podľa potreby dokážeme spracovať užívateľov, modely a diagramy. Tieto moduly uchovávame v databáze, čím sme splnili ich spracovanie podľa potreby.

Vytvorili sme podporné knižnice v jazyku C#, aby sme rozdelili problém na menšie komplexné a samostatné časti, čím sme dostali samostatnosť modulov. Hlavným programom je webová aplikácia, ktorá má svoje rozhranie prispôsobené na modelovanie. V rámci priebehu riešenia od vytvorenia diagramu až po jeho validáciu využívame rôzne vytvorené typy diagramu, presnejšie triedy, čím sme dosiahli oddelenie modelovacej časti od štruktúrovanej časti v programe. Zachytávame všetky namodelované časti a dokážeme ich využiť aj v programe.

Predtým, ako začne samotný proces modelovania, je potrebné uchovávať aj užívateľov systému. Každý diagram má svoj model pravidiel a konkrétneho užívateľa. Pri vytváraní sa zvolí názov diagramu, ktorý bude zachytávať modelovaný proces. Za povinné parametre sa považujú aj výber modelu a užívateľa.

Počas modelovania má užívateľ možnosť overovať si diagram priebežne. Výsledok overovania pravidiel ontológie sa mu zobrazí v modálnom okne, ktoré sa zobrazí po vyhodnotení procesu overovania. Zahŕňa v sebe hlášky, aké algoritmom overovania vyhodnotil program pri vyhodnocovaní. Pre validáciu sme vytvoril algoritmus, ktorý pokryje celý diagram, všetky jeho body. Tento algoritmus nám vyhotoví hlavnú aj menšie vedľajšie cesty, ktoré spoločne slúžia k overovaniu pravidiel ontológie. Taktiež je tu i možnosť exportovania diagramu v podobe JSON súboru, a to buď pre procesy modelovania alebo ako podklad pre analýzu proces miningu.

Zadanie a ciele práce považujem za splnené, pretože sme dokázali splniť kladené požiadavky. Vytvorili sme webovú aplikáciu v technológií .NET Core. Riešením sú aj podporné knižnice, ktoré majú v sebe mapovanie pravidiel ontológie, modelu modelovania, generátor diagramu, algoritmus pokrytia diagramu, vytvorenie modelu diagramov v štruktúre diagramu.

Za prínos považujem spoluprácu s Bc. Miroslavom Babralom, s ktorým sme pravidelne konzultovali postup riešenia tak, aby dokázal včas nadväzovať na výsledky modelovania vo svojej časti, ktorá sa zaoberá procesom miningu. Webová aplikácia nesie názov Uml Business Modeler. Jej výsledkom je diagram v súbore JSON, ktorý využíva Miroslav vo svojom riešení diplomovej práce.

Nápomocnou mi bola aj práca v novovzniknutej multiplatformovej technológii .NET Core, ktorá má v sebe špecifické komponenty, ktoré som sa musel naučiť používať. Napomohlo mi i štúdium metód modelovania procesov a ich možností modelovania. Knižnica GoJS bola spočiatku z hľadiska pochopenia zložitá, ale postupným štúdiom dokumentácie a manuálnym prechádzaním tutoriálov, manuálov a návodov sa moje zručnosti zlepšovali.

Aplikácia má svoje využitie pre podniky, ktoré chcú vyzualizovať svoje procesy pomocou diagramu aktivít. Obsahuje meta-model, ktorý poskytuje zachytávanie vlastností procesu. Pomocou vyhotovených pravidiel je možné namodelovaný proces upravovať podľa vlastných požiadavok, pravidlá ontológie však musia zostať zachované.

Aplikácia by mala rizsiahlejšie využitie, keby sa v budúcnosti zamerala na implementovanie ďalších UML diagramov, čím sa zvýšila možnosť využitia.

6 Záver

Táto diplomová práca bola primárne zameraná na vykonanie výskumu podnikových procesov v rámci podnikového modelu. Výskumom sme zistili, že podnikový model obsahuje všetky procesy v podniku spolu so špecifickými komponentami.

Cieľom práce bolo vytvoriť webovú aplikáciu, ktorá bude slúžiť ako nástroj na modelovanie podnikových procesov a jeho spätné overovanie.

Modelovanie sme dokázali spracovať použitím moderného frameworku GoJS. Vytvorili sme vlastnú knižnicu, ktorá využívala tento framework, a dosiahli sme potrebnú funkcionalitu modelovania.

Overovanie biznis procesov pozostáva z modelu ontológie. Tento model obsahuje väzby, uzly a závislosti, ktoré musí konkrétny biznis proces spĺňať. Taktiež musí obsahovať preddefinované uzly a nadväznosti v diagrame. V práci sme vizuálne oddelili prvky ontológie od ostatných častí modelovania vďaka použitiu špeciálnych znakov, hranatých zátvoriek a mriežky.

Pre overenie procesov bol vytvorený algoritmus, ktorý zachytáva celú štruktúru diagramu. Modelované diagramy je možné exportovať pre účely proces miningu, ktoré využil spolužiak Bc. Miroslav Babral vo svojej práci. Takisto vytvoril parser na spracovanie pravidiel zo súboru ontológie.

Úvodná kapitola podáva základné informácie o rozoberanej problematike. Sú tu analyzované pojmy ako biznisový model - čo reprezentuje a z čoho sa skladá podnikový proces. Dôležitým krokom bolo vysvetlenie biznisového procesu, keďže tvorí podstatnú časť danej problematiky. Predviedli sme tu dve formy modelovania abstrakcií pomocou semi-formálnych a formálnych metód. Jazyk UML, ktorý sa využíval pri danom riešení, je ku koncu tejto kapitoly bližšie rozobraný. Nachádzajú sa tu spomenuté funkčné, štrukturálne a chovacie prístupy. Sú tu taktiež aj typy diagramov.

Po rozbore problému bol v nasledujúcej kapitole opísaný detailný vývoj aplikácie. Vývoj sa viedol celým postupom - od špecifikácií požiadaviek, analýzy problému, návrhu riešenia, implementácií cez testovanie až po nasadenie aplikácie. V špecifikáciách požiadaviek sme rozobrali ciele tejto práce. V rozbore sme prehodnotili analýzu riešenia, spravili sme prehľad funkcionality diagramu aktivít, architektúry aplikácie. Pri návrhu aplikácie sme predviedli návrhy riešenia, ktoré zahŕňali užívateľské rozhranie pre modelovanie, vybratie komponentov meta-modelu, správu užívateľov, diagramov a modelov ontológie. Analyzovaný tu bol proces modelovania, databázový systém a algoritmus overovania procesov. V implementácii boli prezentované použité technológie a štruktúra riešenia softvéru. Pri testovaní sme predviedli príklad riešenia modelovania a overovania. Pri nasadení aplikácie sme opísali architektúru a komponenty, na ktorých je aplikácia nasadená.

Podrobný popis modelovania bol obsiahnutý vo štvrtej kapitole, ktorá opisuje vytvorenú knižnicu UmlDiagramModeler.js, ktorá je založená na modelovacej knižnici GoJS framework.

V ďalšej kapitole boli vylíčené experimenty, ktoré sa prevádzali pri vytvorení nášho riešenia. Sú tu experimenty, ktoré viedli postupnými krokmi a testovaním riešenia takým smerom, aby bola práca dokončená včas a s požadovaným cieľom modelovania podnikových procesov.

Posledná, piata kapitola predstavuje vyhodnotenie práce. Zahŕňa výsledky riešenia, hodnotí splnenie cieľov, ktoré boli na začiatku práce stanovené.

Literatura

[1] Business Modeling with UML, Business Patterns at Work; Autor = Hans-Erik; Penker, Magnus Eriksson, Publisher = Wiley Computer Publishing, Year = 2000, ISBN = 0471295515, URL = https://www.amazon.com/Business-Modeling-UML-Patterns-Work/dp/B006MNAPNQ?SubscriptionId=0JYN1NVW651KCA56C102tag=techkie-20linkCode=xm2camp=2025creative=165953creativeASIN=B006MNAPNQ

[2] Softwarové inženýrství, Author = Sommerville Ian, Publisher = Computer Press, year = 2013, address = Brno, isbn = 978-80-251-3826-7

[3] Introduction to business modeling using the Unified Modeling Language (UML), *IMB* [online], Autor = Jim Heumann, Publisher = IBM, USA: [cit. November 18, 2003]. Dostupné z: https://www.ibm.com/developerworks/rational/library/360.html.

[4] 10 JavaScript libraries to draw your own diagrams, *modeling-languages.com* [online], Autor = Hamza Ed-Douibi, Publisher = modeling-languages.com, [cit. 3. april 2014]. Dostupné z: https://modeling-languages.com/javascript-drawing-libraries-diagrams/

[5] Top online UML modeling tools in 2017 (also including web-based tools for ER and BPMN diagrams), *modeling-languages.com* [online], Autor = Hamza Ed-Douibi, Publisher = modeling-languages.com, [cit. November 18, 2003]. Dostupné z: https://modeling-languages.com/web-based-modeling-tools-uml-er-bpmn/.

[6] Top 5 : Best free diagrams javascript librarie,s *ourcodeworld.com* [online], Autor = Carlos Delgado, Publisher = ourcodeworld.com, [cit. Máj 30 , 2016]. Dostupné z: https://ourcodeworld.com/articles/read/159/top-5-best-free-diagrams-javascript-libraries.

[7] UML Activity Diagrams: Guidelines, *tutorialspoint.com* [online], Autor = Nirbhay Singh, Publisher = Microsoft, [cit. Augutst 18, 2015]. Dostupné z: https://www.tutorialspoint.com/uml/uml_activity_diagram.html.

[8] Business model, *Microsoft* [online], Autor = Microsoft, Publisher = MSDN, USA: MSDN [cit. November 18, 2003]. Dostupné z: https://msdn.microsoft.com/en-us/library/dd409465.aspx.

[9] GoJS Framework, *Northwoods Software* [online], Autor = Northwoods Software, Publisher = Northwoods Software, [cit. 1998-2018]. Dostupné z: https://gojs.net/latest/index.html.

[10] Get started with .NET in 10 minutes, *Microsoft* [online], Autor = Microsoft, Publisher = Microsoft, [cit. Februar 12, 2012]. Dostupné z: https://www.microsoft.com/net/learn/get-started/windows.

[11] Google Cloud, *Google* [online], Autor = Google, Publisher = Google, [cit. Januar 11, 2015]. Dostupné z: https://cloud.google.com/why-google-cloud/.

[12] Disrupting the Business of Diagramming, *draw.io* [online], Autor = Gaudenz Alder, Publisher = draw.io, Northampton: [cit. Marec 8, 2014]. Dostupné z: https://about.draw.io/about-us/.

[13] How to Draw Event-Driven Process Chain Diagram (EPC Diagram)?, [online], Autor = visual-paradigm.com, Publisher = visual-paradigm.com, [cit. September 25, 2017]. Dostupné z: http://www.scholarpedia.org/article/Petri_net.

[14] Petri Net, *Scholarpedia* [online], Autor = Carl Adam Petri and Wolfgang Reisig, Publisher = Scholarpedia, USA: Hamburg [cit. 2008]. Dostupné z: https://www.ibm.com/developerworks/rational/library/360.html.

[15] ASP.NET Core MVC with EF Core - tutorial series [online], *Microsoft*, Autor = Microsoft, Publisher = Microsoft, [cit. April 21, 2017]. Dostupné z: https://docs.microsoft.com/cs-cz/aspnet/core/data/ef-mvc/.

[16] Ontology Development 101: A Guide to Creating Your First Ontology [online], *Stanford University*, Autor = Ontology Development 101: A Guide to Creating Your First Ontology, Publisher = Stanford University, Kanada [cit. September 2011]. Dostupné z: https://protege.stanford.edu/publications/ontology_development/ontology101-noy-mcguinness.html.

A Obsah priloženého CD

Piložené CD obsahuje zabalený súbor vo formáte .zip s riešením diplomovej práce. Obsahuje všetky vytvorené komponenty. K riešeniu je priložený textový súbor s návodom na spustenie webovej aplikácie. Súbor obsahuje vytvorené súbory ontológie pomocou programu Protégé, ktoré sa nachádzajú v zložke TestData. Nachádza sa tu ešte jeden textový súbor, v ktorom je odkaz na nasadenú aplikáciu na Google Cloude.